GEM *Programmer's Toolkit* ™
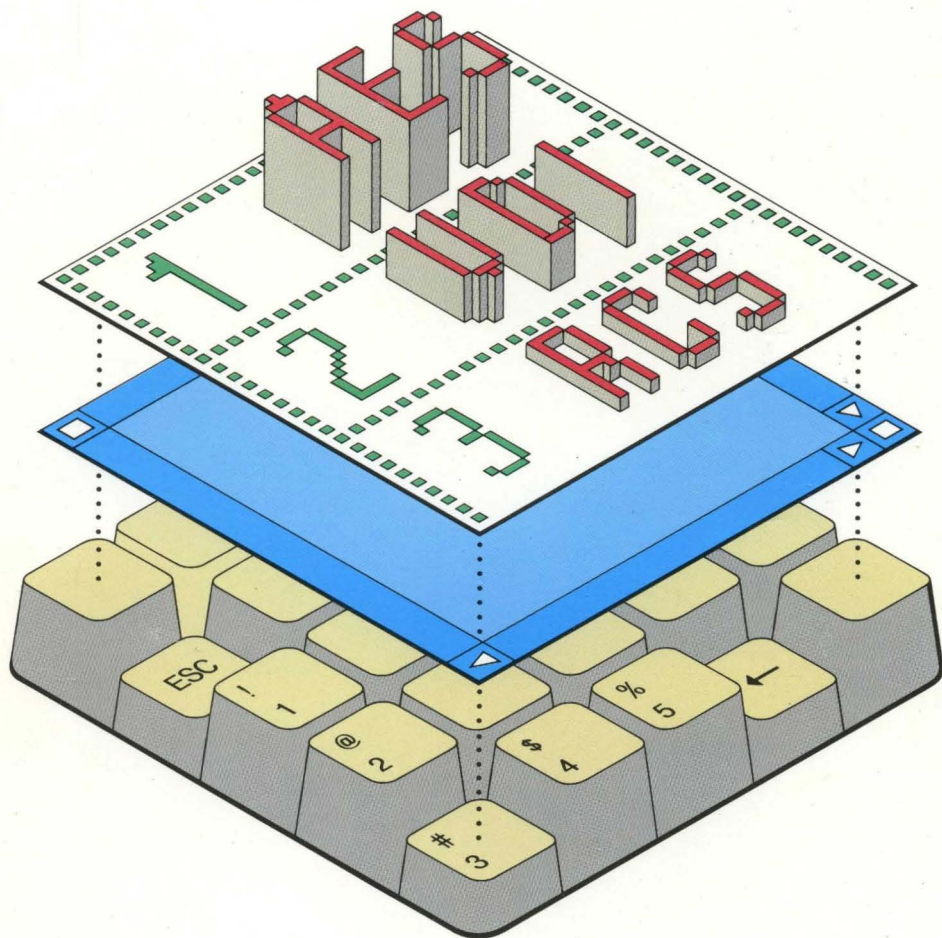
Vol 3
GEM™ RCS

GEM™ Resource Construction Set

Release Note 01
GEM Programmer's Utilities Guide
First Edition, June 1986

Converting .DEF Files in the 68K Environment

In the 68K environment, to convert an existing .DEF file to the .DFN format used by GEM^{TM} Resource Construction Set, you cannot simply rename the file. You must run the utility program DEF2DFN.PRG.

Programming for the Atari^{TM} ST

If you are writing an application for the Atari ST series, note the following:

● The GEM software, including GEM RCS, does not support the Atari's 40-character low-resolution mode.

● Because the Atari TOS^{TM} operating system does not force pathnames or filenames to upper case, certain fields (notably the "Object Name" field in several GEM RCS dialogs) require that you enter the characters in upper case. If you use lowercase characters, nothing appears in the field.

● To allow the ST to use a television set as a monitor, the length of the Message Line in an ALERT tree (GEM RCS) is restricted to <u>thirty</u> characters, not the forty characters supported by other DOS and 68K systems. GEM RCS does not enforce this restriction; the responsibility is yours. However, GEM RCS helps you by including character counts in the Edit Unformatted String Object dialog.

End of Release Note

Contents

Introduction to GEM RCS

GEM^{TM} Resource Construction Set (RCS) is a GEM application you use
to create resources (menus, dialog boxes, alert boxes, etc.) for your
GEM application programs. GEM RCS also lets you incorporate icons
and bit-images you create with GEM IconEdit into your resource files.

You don't have to be a programmer to use GEM RCS. You can create
all of the application's resources and then give the file to a
programmer to include with the program code.

This document is divided into the following sections:

● Section 1, **Introduction to GEM RCS**, explains what a resource is,
  describes the GEM RCS screen, discusses the mouse techniques you
  use with GEM RCS, and describes the GEM RCS workspace and how
  GEM RCS uses memory.

● Section 2, **GEM RCS Tutorial**, acquaints you with basic terminology
  and techniques as you construct a menu and a dialog box.

● Section 3, **GEM RCS Reference**, contains detailed descriptions of
  GEM RCS's features, including the parts box, the objects you can
  use in your resources, the clipboard, toolkit, and menus.

● Section 4, **RSCREATE and Other Technical Information**, describes
  RSCREATE, a utility program with which you can modify and port
  resource files. The section also describes how to chain resource
  files and how to deal with unknown object tree types, and provides
  some tips for speedy and efficient use of GEM RCS.

Many of the techniques you use with GEM RCS, such as choosing a
command from a drop-down menu, are the same as those you use
with the GEM Desktop^{TM}. Be sure you have read your <u>GEM Desktop</u>
guide and understand these basic operations before you start working
with GEM RCS.

WHAT ARE RESOURCES?

In GEM applications, resources are things that appear on the screen
(like menus, dialogs, or alerts) that are not actually part of the program
code. Instead, resources are kept in a separate resource file, an
arrangement that has several advantages.

● As noted previously, the resource file can be created by a non-
programmer.

● The resource file can be modified or updated (again by a non-
programmer) often without the application code having to be
recompiled.

● An application can exist in different national "editions," using the
same program code and different resource files for each nationality.

● An application can operate in different machine environments, using
resources generated from the same source code.

Resources are made up of objects. "Object," in this sense, is a
technical term referring to a specific set of images that can appear on
the screen, including empty boxes, boxes containing text, text strings,
and the like.

To create a resource--a menu, for example--you combine objects to
form an "object tree." The relationship between the objects in the tree
is described in family terms: the first object is the "parent"; the objects
contained within the parent are the "children."

For a complete description of objects and object trees, see Section 6,
"Object Library," in the GEM Application Environment Services
Reference Guide. (The GEM Application Environment Services are
referred to as GEM AES.)

Using GEM RCS, you can create the following kinds of object trees:

MENU   The drop-down menus characteristic of GEM applications
are contained in the resource file. No menu can be
more than one-fourth the size of the screen. This
makes it possible for the part of the screen covered by
the menu to be written to a special buffer (see the
discussion of the menu/alert buffer in the Introduction to

GEM Programming) and for the screen to be redrawn by the GEM AES from this buffer. A screen redraw from the menu/alert buffer is faster than a redraw handled by the application.

DIALOG

To the end-user, a dialog is a GEM application's means of providing or getting information, but GEM applications can use dialogs for other purposes. For example, the GEM RCS toolkit (see Figure 1-1) is actually a DIALOG in the resource file.

Dialogs do not use the menu/alert buffer and thus can be larger than one-fourth the size of the screen. The objects in a dialog "snap" to character boundaries on the screen; this snap makes it easier to align the elements of the dialog.

PANEL

A panel is similar to a dialog, except that it does not have the automatic character boundary snap. A panel can be used for anything that requires precise positioning of objects. For example, the GEM RCS parts boxes (see Figure 1-1) are panels in the resource file.

ALERT

Alerts (a specialized subset of dialogs) are a GEM application's means of displaying notices, warnings, or error messages. Alerts have a fixed format and a size limit of no more than one-fourth the size of the screen. (They use the menu/alert buffer mentioned previously.) The alert format includes an icon, a 200-character ASCII text string (five lines of no more than forty characters each), and up to three exit buttons. Each exit button has a twenty-character maximum.

FREE

This category includes strings and bit-images you want to include in the application, but not as part of the code. In this way, you can change the string or bit-image without having to change the code and recompile.

One example of a free string is the non-default state of a context-sensitive menu command. For example, GEM RCS's Global Menu has a command whose default state is **Hide Parts**. The default version of the menu is contained in the MENU tree; the non-default state of the command (**Show Parts**) is stored as a free string.

GEM RCS SCREEN

When you start GEM RCS, your screen looks like the illustration below. The labeled components are described after the illustration.
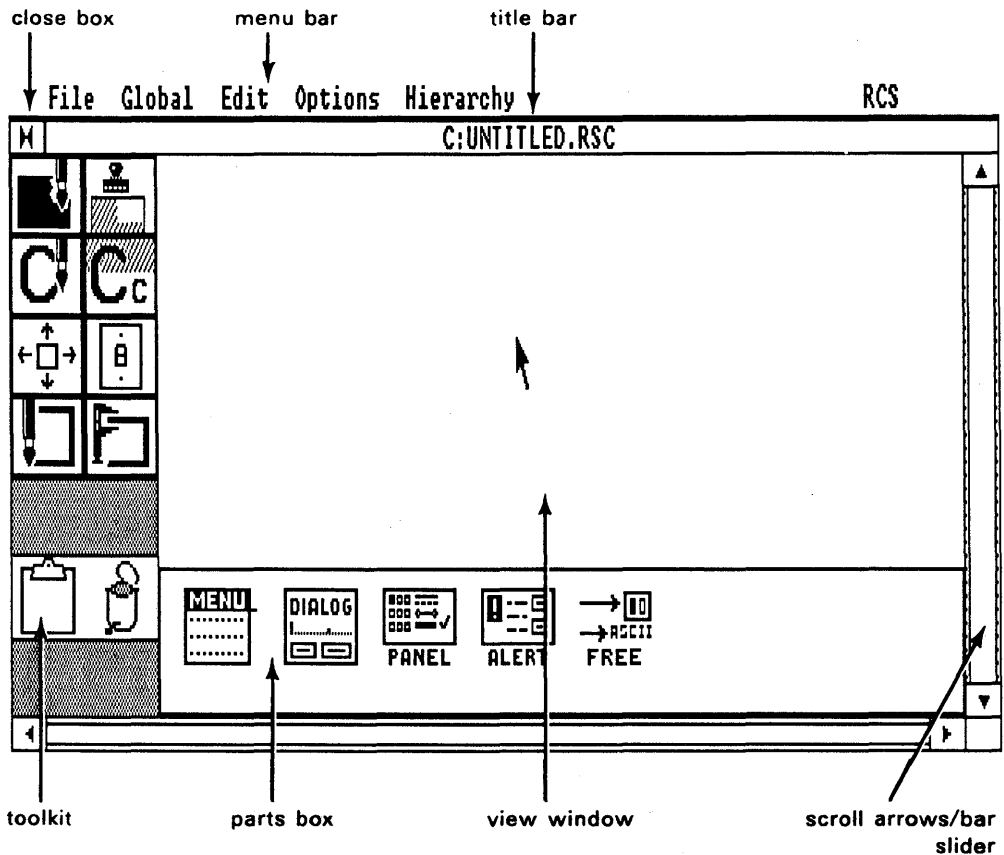
Figure 1-1.   GEM RCS Screen

Menu Bar

The menu bar is the top line of your screen. It contains the menu titles File, Global, Edit, Options, Hierarchy, and RCS.

When you touch a menu title with the pointer, the menu drops down below the menu bar. You can then choose a command by clicking on its name in the menu.

GEM RCS's menu commands are described in detail in Section 3. Most of the commands can also be executed by typing one of the keystroke combinations listed in Table 3-4.

Title Bar

The title bar shows which of GEM RCS's two levels you're working in:

- root level
- object level

The root level is the highest level of the resource file. At this level, the title bar identifies the current directory path and the resource filename--**C:\TOOLS\MYAPP.RSC**, for example--and the view window contains the icons for the object trees (MENU, DIALOG, etc.) you have placed in the resource file.

The object level is the level at which you actually work on the objects in a tree--for example, adding text strings, command entries, or exit buttons. At this level, the title bar contains the tree's name (FILEMENU, for example), without any directory path information.

Note the following about the name of your resource file:

- Each resource file starts out as **UNTITLED.RSC** until you save it and give it a name.

- The resource file extension is .RSC (resource). Don't confuse it with the application's name: GEM RCS (Resource Construction Set).

Close Box

Clicking on the close box does the following:

● If you are at the object level, the close box closes the object and returns you to the root level of the resource file.

● If you are at the root level, the close box closes the resource file. If you have edited the file but not saved the edits, GEM RCS displays a dialog asking if you want to abandon the edits or save the file.

Note: If you are working on a new file that hasn't yet been named and saved, the close box has no effect at the root level.

The close box and the **Close** command on the File Menu can be used interchangeably.

Toolkit

The toolkit (described in detail in Section 3) contains icons for the various tools you can use to "customize" the objects in a tree. For example, you can change an object's color, fill pattern, or alignment with the tools.

View Window

The view window is where you design the layout of a tree and customize its objects.

Parts Box

The parts box (described in detail in Section 3) contains icons for the objects you can include in an object tree. The contents of the parts box change according to the type of tree you're working on.

Scroll Bars and Sliders

GEM RCS's scroll bars and sliders work in the same manner as the scroll bars and sliders on GEM Desktop windows. Note that in most cases you can scroll horizontally as well as vertically.

MOUSE TECHNIQUES

Many mouse techniques you use with GEM RCS, such as clicking and dragging, are the same as those you use with the GEM Desktop and other GEM applications. However, the effects of these techniques are often specific to GEM RCS, as the following table illustrates:

Table 1-1. GEM RCS Mouse Techniques

Technique	Effect
click	Selects tree or object. GEM RCS indicates a selected tree by highlighting its icon in reverse video and a selected object by defining its "extent" with a dotted line.
Shift-click	Selects more than one object.
Ctrl-click	Selects parent of object.
double-click	Opens tree or object for editing.
drag	Copies tree or object from parts box to view window; moves tree or object inside view window; moves tree or object to or from clipboard.
Ctrl-drag	Moves object's parent and all that parent's children.
Shift-drag	Copies tree or object inside view window; copies tree or object to or from clipboard.
Ctrl-Shift-drag	Copies object's parent and all that parent's children.

Note: When using any of the dragging techniques to move or copy trees or objects, remember the following:

● If you are copying from the parts box or if you are moving or

copying within the view window, the dotted line extent of the tree or object must be entirely inside the view window or a parent object. Here are two specific examples:

- Dragging a tree icon from the parts box to the view window: if any part of the extent is in the title bar or the toolkit when you release the mouse button, the tree is not copied to the view window.

- Dragging a MENU tree ENTRY from the parts box to the menu box in the view window: if any part of the ENTRY extent is outside the menu box (its parent) when you release the mouse button, the ENTRY is not copied to the menu box.

● If you are moving or copying a tree or object to the clipboard, drag from the upper left corner of the tree or object. If you don't, and if any part of the extent is off the left edge of the toolkit, the tree or object is not moved or copied to the clipboard. (The clipboard is described in Section 3.)

Note also that when you copy a tree or object, the copy does not retain the tree or object names associated with the original.

Sizing Objects

To change the size of an object, first select the object. GEM RCS highlights the selected object with a dotted outline and displays the object's "size handle" (a solid black rectangle) in the bottom right corner. Then place the pointer on the size handle and drag the size handle until the object is the size you want.

If the object has children, note that GEM RCS won't permit you to make a parent object too small to contain its children.

Deleting Trees or Objects

To delete a tree or object, do any of the following:

● Drag it from the view window to the trash can icon in the toolkit. When the trash can icon is highlighted, release the mouse button.

● Click on the tree or object to select it. Then move the pointer to

the trash can icon (without dragging the tree or object), and click on the trash can.

● Select the tree or object and then choose the **Delete** command from the Edit Menu.

You cannot retrieve an item once you place it in the trash can.

WORKSPACE AND MEMORY USE

When you start GEM RCS, it requests a portion of your computer's memory as its workspace. Depending on how much RAM is available, this workspace can be as much as 64K.

As you add trees and objects, you whittle away at the available workspace. However, if you delete a tree or object, GEM RCS does not restore the portion of the workspace occupied by the tree or object. The only way you can regain that memory is by saving the file. The **Save** and **Save As...** commands (described fully in Section 3) write out and reload the resource.

To find out how much of the workspace is available, choose the **Info...** command from the Options Menu. (Menu commands are described fully in Section 3.) The information dialog tells how many bytes the file, tree, or object has used and how much space remains in the workspace.

Note: The file size shown in the information dialog is only an approximation of the size of the resource file GEM RCS will produce, for the following reasons:

● Alerts actually take less space in the final file because they are written out as ASCII strings.

● Shift-dragged copies of objects might use more space in the final file.

End of Section 1

GEM RCS Tutorial

In this tutorial introduction to GEM RCS, you will start GEM RCS and then create the menu and dialog box shown below.

```
┌─────────────────────┐
│ Options             │
├─────────────────────┤
│ Load File...    ^L  │
│ Delete File...  ^D  │
│ ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓  │
│ Rotate Symbols      │
└─────────────────────┘
```

```
┌────────────────────────────────────────────────────┐
│                                                    │
│   Political party?  Democratic  Republican  Independent │
│                                                    │
│                                                    │
│              ┌──────────┐  ┌──────────┐            │
│              │   OK     │  │ Cancel   │            │
│              └──────────┘  └──────────┘            │
│                                                    │
└────────────────────────────────────────────────────┘
```
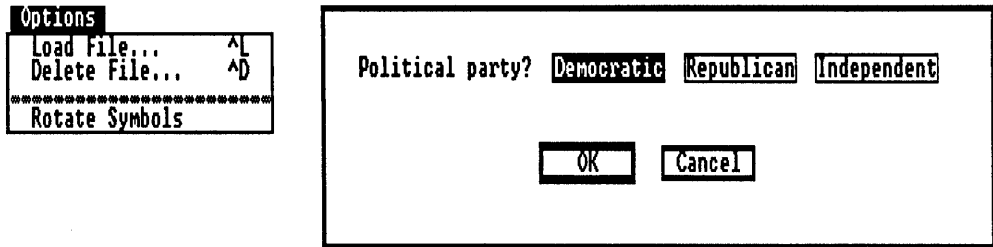
You will then save the resource file and exit GEM RCS.

STARTING GEM RCS

To start GEM RCS, start the GEM Desktop and then do the following:

1. Open the TOOLS folder and locate the RCS.APP icon.
2. Double-click on the RCS.APP icon.

CREATING A MENU

The menu you will create in the following steps contains simple text string commands. This part of the tutorial demonstrates opening a tree, adding objects to the tree, setting attributes, and sorting objects.

Choosing and Naming a MENU Object Tree Icon

The parts box contains icons for the five kinds of object trees you can construct with GEM RCS:

Drag the MENU icon into the view window. You don't have to locate the icon precisely; when you release the mouse button, GEM RCS automatically places a copy of the icon in the upper left corner of the window. (Remember that GEM RCS does not copy the icon to the view window if any part of the outline you're dragging is in the toolkit or the title bar.)

Whenever you place a new object tree in the view window, GEM RCS displays a dialog that asks you to name it.
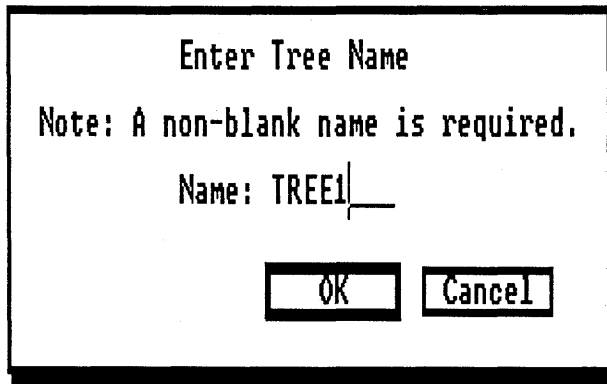
Note that the dialog already contains the name TREE1. To use this name for your object tree, you would simply click on the dialog's OK button or press the Enter key. In this case, however, give the object tree a different name. Press the Esc key to erase TREE1, type the name **APPMENUS**, and then click on the OK button or press the Enter key. (See your GEM Desktop guide for a full description of entering and editing text in dialogs.)

Opening the Object Tree

Double-click on the APPMENUS icon. The following changes take place:

● The title bar now says APPMENUS.

● The view window contains a menu bar with two titles--File and Desk--and a shaded background area. (The Desk Menu is at the extreme right of the menu bar.)

● The parts box contains the four objects that can appear in the MENU object tree: the menu TITLE, an ENTRY (each command text string is an ENTRY), a line pattern to separate entries, and a box in which you can place menu items that are not text strings (fill patterns or line styles, for example).

In this tutorial you'll create a new menu from scratch. In Section 3 we'll tell you how you can use the existing File Menu and Desk Menu.

Adding a Menu Title

Drag the TITLE object from the parts box to the menu bar, placing it after "File". Be careful not to drag into the GEM RCS title bar or the toolkit. If you do, GEM RCS cancels what you've dragged.

Now double-click on TITLE. The following dialog appears on your screen:

```
         Edit Unformatted String Object

Text: STRING|_____
                   ^        ^        ^
                   20       30       40 characters

(Optional) Object Name: _____     [  OK  ]  [Cancel]
```

Do the following to edit and name the menu's title:

1. Press the Esc key to erase the text.

2. Press the spacebar, type **Options**, and then press the spacebar again. (If you don't press the spacebar, there won't be enough blank space between menu titles.)

3. Move the cursor to the Object Name field and type **OPTITLE**.

4. Click on the OK button or press the Enter key.

Adding Entries to the Menu

To add entries (commands) to the menu, do the following:

1. Click on the menu's title, Options. This selects the title and also displays a small menu box below the menu bar.

2. Place the tip of the pointer just inside the lower right corner of the menu box and press the mouse button. When the pointing finger icon appears, drag down and to the right to make the menu box larger. Make it larger than you think you need; you'll make it the right size later.

3. Drag an ENTRY from the parts box and place it in the upper left corner of the menu box. Don't crowd the corner too tightly, and make sure the outline you're dragging is entirely inside the menu box.

4. Copy this ENTRY by Shift-dragging. Place the copy just below the first ENTRY.

5. Drag a separator line from the parts box and place it just below the two entries.

6. Shift-drag the ENTRY once more, placing the new copy below the separator line.

Editing the Entries

The next step is to convert each generic ENTRY into a command an end-user can choose. To do so, do the following:

1. Double-click on the first ENTRY in the menu. GEM RCS displays the Edit Unformatted String Object dialog.

2. Press the Esc key to erase the text in the dialog.

3. Press the spacebar <u>twice</u> (to put two blank spaces before the command) and then type **Load File....** (The three dots are a GEM convention indicating that choosing the command will cause a dialog to be displayed.) Don't click on the OK button yet.

4. Many GEM applications use keystroke equivalents for their menu commands. Let's say the end-user could type Ctrl-L to produce the same effect as choosing the **Load File...** command. To indicate this in the menu, press the spacebar several times and then type ^ L, followed by a single blank space.

5. Move the cursor to the Object Name field and type **OPSLOAD**.

6. Click on the OK button or press the Enter key.

Note to Step 4: GEM application menus follow the convention that ^ represents the Ctrl key and that a filled diamond represents the Alt key. (The keystroke combination **Ctrl-G** produces a filled diamond in the Edit Unformatted String Object dialog.) GEM application menus also follow the convention of surrounding the command text string with two leading blank spaces and a single trailing blank space.

Take the same steps for the second ENTRY, with the following variations:

● The command is **Delete File....**
● The keystroke equivalent is ^ **D**.
● The Object Name is **OPSDELT**.

Don't forget the leading and trailing spaces, and don't worry if the ^ **L** and ^ **D** don't line up properly at first. You can fix that simply by adding or removing spaces in the Edit Unformatted String Object dialog. The editing techniques are the same as for any GEM application dialog.

Now edit the third ENTRY, with the following variations:

● The command is **Rotate Symbols**.
● It has no keystroke equivalent.
● The Object Name is **OPSROTE**.

Sizing the Menu Box

To make the menu box the right size, first select the box by clicking inside it away from the commands or the separator line. You'll know you have been successful if the size handle appears at the box's lower right corner.

Next, drag the size handle until the menu box is the size you want.

You can also size the menu box without first selecting it. Use the same technique described in Step 2 under "Adding Entries to the Menu."

Sizing the Commands

Because GEM AES only highlights and enables the area contained within the extent of an ENTRY, you should make each command's extent match the full width of the menu box. If you don't, your application's menus will have a visual and functional inconsistency that the end-user will probably find displeasing.

To size a command, you can do any of the following:

● Select it and drag its size handle all the way to the right border of the menu box.
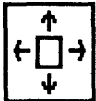
● Select it, display the alignment pop-up menu (see left), and choose the **Fill Horizontal** option.

● In the Text field of the Edit Unformatted String Object dialog, put blank spaces after the string value of each ENTRY.

Extending the Separator Line

To extend the separator line all the way across the menu box, do the following:

1. Double-click on the separator line. GEM RCS displays the Edit Unformatted String Object dialog. The heavy bars in the Text field of the dialog are equivalents to the shaded separator line in the menu.

2. To extend the line, type several Ctrl-S combinations in the Text field.

3. Click on the OK button or press the Enter key.

If the line is not long enough, repeat these steps until it is. If the line is now too long, GEM RCS displays an alert that this object no longer fits inside its parent. Click on the dialog's OK button and then double-click on the separator line again. Use the Backspace key to erase the bars until the separator line is the right length.

Sorting the Objects in the Menu

The last thing you should do before closing a tree is sort the objects in it. If you don't, at run-time your application will draw the objects at each level of the tree in the order in which you created them, which may be visually displeasing. By sorting the objects, you can determine the order in which they will be drawn.

To sort the objects in your menu tree, do the following:

1. Ctrl-click on any of the commands. This selects the menu box.

2. Display the Hierarchy Menu and choose the **Sort Children...** command. GEM RCS displays a dialog with four sorting options: single-row, single-column, double-column, double-row.

3. Choose the single-column option (second from the left) and press the Enter key or click on the OK button.

Closing the MENU Object Tree

Your menu is now complete. To close its object tree and return to the root level of the resource file, click on the close box or choose the **Close** command from the File Menu.

CREATING A DIALOG BOX

The dialog you will create in the following steps contains a text string, a set of "radio buttons," and two exit buttons. In addition to demonstrating more about setting attributes and sorting objects, this

part of the tutorial illustrates working with a three-level tree, where one of the children of the root object has children of its own.

Choosing and Naming a DIALOG Object Tree Icon

The first step in creating a dialog box is the same one you took in creating the menu: choosing and naming the object tree icon.

Drag the DIALOG object icon from the parts box and place it in the view window. When GEM RCS displays the naming dialog, press the Esc key to erase the default name (note that it is TREE2; the MENU was TREE1), type **PTYDIAL**, and click on the OK button or press the Enter key.

Opening the Object Tree

To open your dialog box's object tree, double-click on the PTYDIAL icon. Note the following changes to your screen:

● The name of your object tree, PTYDIAL, appears in the title bar at the top of the view window.

● The view window is now empty.

● The icons in the parts box change, showing you the objects you can include in your dialog box.

Choosing Objects for Your Dialog Box

To assemble the objects for your dialog box, do the following:

1. Drag the STRING object from the parts box, placing it in the upper left of the view window.
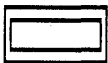
2. Drag the hollow box object (see left) from the parts box and place in the upper center of the view window.

3. Click on the box to select it. Note the object's size handle (mentioned under "Sizing Objects" in Section 1) at the lower right corner of the extent.

4. Drag the box's size handle down a little and almost to the right border of the view window, making a long, shallow rectangle.

5. Drag the BUTTON object from the parts box and place it inside the box you just enlarged, in the upper left corner.

Customizing the STRING Object

Double-click on the STRING object in the view window. When GEM RCS displays the Edit Unformatted String Object dialog, press the Esc key to erase the text and then type the following:

Political party?

You don't need to do anything with the optional object name, so click on the OK button or press the Enter key to complete your work in the Edit Unformatted String Object dialog. The STRING object in your dialog box now contains the phrase you just typed. (Don't be concerned if the message overlaps the box or BUTTON object. We'll reposition everything later.)

Customizing the BUTTON Object

To change the text inside the BUTTON object and to give it a name to which you can refer in your program code, do the following:

1. Double-click on the button. GEM RCS displays the Edit Unformatted String Object dialog.

2. Press the Esc key to clear the existing text.

3. Type **Democratic**.

4. Click on the "Object Name" field. The text cursor moves to this line. (You can also move the cursor by pressing the Tab key or the down-arrow key.)

5. Type **DEMBTN**. (To make reading your code as easy as possible, you should use object names that indicate the object's function.)

6. Click on the OK button or press the Enter key.

Setting a Radio Button

The "Democratic" button and two others (you'll create them in just a moment) will make up a set of "radio buttons." Like the pushbuttons on a car radio, radio buttons have the following characteristics:

● In a set, only one button at a time can be selected, but there should always be a selected button. There should never be a case where no button is selected.

● Selecting a button automatically de-selects the previously selected button in the set.

To set the button as a radio button, do the following:

1. Click on the "Democratic" button to select it.

2. Click on the attributes menu icon (see left) and click on "Radio Button" in the pop-up menu that appears.

Creating Additional Radio Buttons

This dialog has two additional radio buttons, "Republican" and "Independent". To create these buttons, do the following:

1. Shift-drag the "Democratic" button twice, placing the two copies to the right of the original.

2. Double-click on the copy immediately to the right of the original. In the Edit Unformatted String Object dialog, change the text to "Republican" and the object name to "REPBTN".

3. Double-click on the last copy. In the Edit Unformatted String Object dialog, change the text to "Independent" and the object name to "INDBTN".

Designating the Selected Radio Button

When the dialog appears at run-time, one of the radio buttons must be pre-selected (highlighted in reverse video). To pre-select the "Democratic" button, do the following:

1. Click on the button to select it.
2. Click on the attributes menu icon and choose "Selected".

Customizing the Parent Box

Before you work on the parent box, you might want to know why you need one in the first place.

When you have more than one set of radio buttons in a dialog, you must enclose each set inside a parent box. If you don't, clicking on one button will turn off <u>all</u> other radio buttons in the dialog, not just the buttons in that one set.

In this dialog, with only a single set of radio buttons, you don't really need a parent box, but the dialog could have additional sets labeled "Age group?" and "Salary range?" This step will demonstrate what to do when you <u>do</u> need the parent box.

First, arrange the radio buttons in the parent box. (The "Democratic" button should be in the box's upper left corner.) Then select the parent box (if it isn't already selected) and size it down as small as you can make it.

You don't want the parent box to be visible in the final dialog. To make it invisible, do the following:

1. Select the box, if it isn't already selected.
2. Click on the border width menu icon (see below).
3. In the border width pop-up menu, choose the null border. See the illustration on the next page.

null border
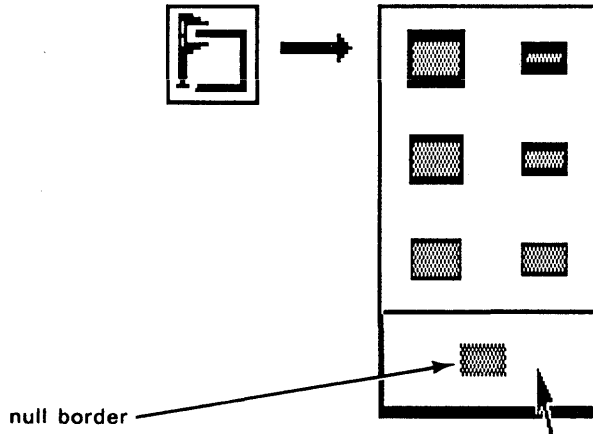
If you de-select the parent box by clicking elsewhere in the view window, you'll see that the parent is now invisible. However, you can still select it by Ctrl-clicking on one of the radio buttons, and you can move it (and the buttons too) by Ctrl-dragging any of the buttons.

Creating Exit Buttons

The dialog needs two "exit buttons" with which the end-user can exit the dialog by clicking on either button.

To create the exit buttons, do the following:

1. Drag a BUTTON object from the parts box, placing it below the radio buttons.

2. Shift-drag this BUTTON object, placing the copy just to the right of the original.

3. Edit and set attributes for the first BUTTON as follows:

    ● Change its text to "OK".
    ● Name it **OKBTN**.
    ● Set its attributes to "Exit" and "Default".

    Making a button the "default" means that at run-time the end-user can select this button either by pressing the Enter key or by clicking

on it. (You've already encountered a default button in the Edit Unformatted String Object dialog: the OK button.)

4. Now edit and set attributes for the second exit button as follows:

   ● Change its text to "Cancel".
   ● Name it **CANCLBTN**.
   ● Set its attribute to "Exit".

5. Finally, locate the exit buttons where you want them.

Resizing the Background of Your Dialog Box

Move the pointer to an open area of the view window (away from the text and buttons) and click. The dialog box's size handle appears in the bottom right corner of the view window.

Place the pointer on the size handle and drag <u>up</u> and to the <u>left</u>. As you drag, the background of your dialog box gets smaller. When the background is the size you want, release the mouse button.

Sorting Objects in the Dialog

The last thing to do before closing this tree is to sort the objects in it. The procedure is essentially the same as with the SAMPLE dialog.

Note: You must sort all levels (sets) of children. For example, this dialog has two levels. The radio buttons are children of the empty box, and the empty box, string, and exit buttons are children of the dialog's outer box.

To sort the children in your dialog, do the following:

1. Ctrl-click on one of the radio buttons. This selects its parent box.

2. Display the Hierarchy Menu and choose the **Sort Children...** command. Click on the single-row option (on the far left) and press the Enter key or click on the OK button.

3. Select the dialog's outer box.

4. Display the Hierarchy Menu and again choose the **Sort Children...** command. This time click on the double-row option (on the far right) and press the Enter key or click on the OK button.

Sorting the objects will draw them in the following order:

1. "Political party?" string.
2. Radio buttons, "Democratic" first.
3. OK button.
4. Cancel button.

Closing the DIALOG Object Tree

Your dialog is now complete. To close its object tree and return to the root level of the resource file, click on the close box or choose the **Close** command from the File Menu.

SAVING A RESOURCE FILE

Display the File Menu and choose the **Save As...** command. The Item Selector appears on your screen.

Type **TEST** and then click on the OK button or press the Enter key. GEM RCS saves the following files:

TEST.RSC   The actual resource file you would include with your application program. Note that you do not need to type the .RSC file extension; GEM RCS automatically supplies it.

TEST.DFN   An auxiliary file that identifies the trees and objects in your resource file. When you work in GEM RCS, you need both the resource file and its .DFN file in the same directory.

        If you have created resource files with GEM RCS, the earlier version of the Resource Construction Set, the definition file has the extension .DEF. To use this file with GEM RCS, simply change its extension to .DFN.

In its initial default configuration, GEM RCS also creates TEST.H, a C language include file you must use when you compile your application program code. The .H file is an ASCII file that lists the trees and objects in your resource file and the object numbers assigned to them

by GEM RCS. You can also read this file to check the order in which objects have been sorted. Sorting is discussed in this tutorial and also in the description of the Hierarchy Menu in Section 3.

Note: You can also produce include files for Pascal, BASIC, and FORTRAN-77, as well as an editable ASCII version of your resource file (the .RSH file). See the description of the **Output...** and **Save Preferences** commands in the Global Menu (Section 3) and the description of RSCREATE in Section 4.

QUITTING GEM RCS

To stop GEM RCS, choose the **Quit** command from the File Menu. You then return to the GEM Desktop.

End of Section 2

GEM RCS Reference

ASPECT RATIO AND SCREEN RESOLUTION

GEM applications can run under a variety of aspect ratios and screen resolutions. The following table lists the three most common combinations.

Table 3-1. Screen Resolution and Aspect Ratio

Resolution	Pixel Ratio	Aspect Ratio
low	640x200	3.2:1
high*	640x400	1.6:1
high*	720x350	2.06:1
square*	any	1:1

*Although their aspect ratios are different, the GEM software uses the same screen driver for these resolutions.

If an icon or bit image is displayed on a system with an aspect ratio different from the system on which it was created, the icon or bit image will appear taller or shorter than it appeared in the original. This can cause problems if you are placing icons or bit images close together; aspect ratio differences can cause them to overlap each other or separate from each other.

You can resolve this issue in either of the following ways:

● You can create a single resource file and compensate for the different aspect ratios of the systems on which you expect your application to run.

● You can create different sets of icons and bit images for each expected aspect ratio. You can then create aspect ratio-specific resource files from a single "master" file by loading the appropriate set of icons and bit images into each final version of the resource file.

The following illustrations demonstrate these two approaches. On the left, you see a partial view of the GEM RCS toolkit, which was created only in a low-resolution version. Note how the bit images are flattened on a high-resolution system. On the right, you see partial views of the GEM Paint^{TM} toolkit, which exists in both high- and low-resolution versions.
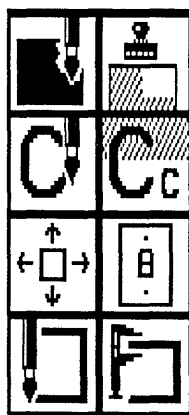
GEM RCS
lo-res
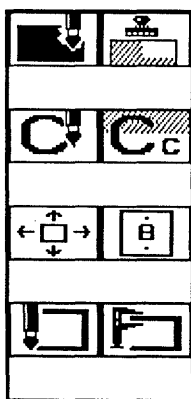
GEM RCS
hi-res
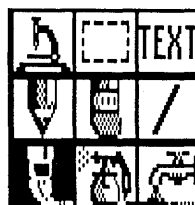
GEM Paint
lo-res
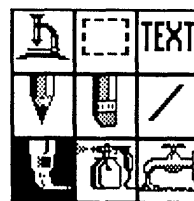
GEM Paint
hi-res

If you create aspect ratio-specific resource files, at run-time your application can determine the screen's aspect ratio and resolution and load the appropriate file.

For more on icons and bit images, see "Icons and Bit Images," later in this section.

Note: Aspect ratio differences do not cause problems with text or any other objects in a tree.

FILE AND DESK MENUS

As noted in the tutorial, when you open a MENU object tree, there are already a File Menu and a Desk Menu in the menu bar.

File Menu

The File Menu is optional; you can drag its title to the trash can.

To add commands to the File Menu, do the following:

1. Click on the menu title. This selects the title and displays the menu box with a single command, **Quit**.

2. Place the pointer at the lower right corner of the menu box, and Ctrl-drag down and to the right. This makes the menu box larger, without affecting the size or location of the command.

3. Drag the **Quit** command to the bottom of the menu box, or place it wherever you want it.

From this point, you can create the menu just as you did in the tutorial.

Desk Menu

This menu is required; you cannot drag its title to the trash. However, at run-time its name and location can vary as follows:

● If your application is running under version 1.X of GEM AES, the menu is called the Desk Menu and appears at the left side of the menu bar.

● If your application is running under version 2.X of GEM AES, the menu appears at the right side of the menu bar, and the menu's name is the filename of the application's executable file. For example, for an application executed by a file called BINGO.APP, it would be the BINGO Menu.

The name variation and menu placement are handled entirely by the GEM AES.

The Desk Menu contains several "placeholder" commands. The first--

Your message here--is fully editable (the message has a 20-character maximum), but you cannot delete it.   A typical function for this command is to display an informational dialog about the application, including its version number.

The numbers below the separator line are placeholders for desk accessories.   The menu can accomodate a maximum of six.   You cannot edit or delete these entries.   GEM AES handles placing desk accessory names in the menu and sizing the menu according to the number of desk accessories it contains.

USING THE PARTS BOX

The parts box contains icons for the objects you can include in an object tree.   These objects have preassigned attributes (fill patterns, colors, text strings, etc.) you can change with the tools in the toolkit. You can edit text in an object or make other changes by double-clicking on the object to display one of a variety of dialogs.

To place an object in your current object tree, drag its icon from the parts box to the view window.

The contents of the parts box depend on the type of tree you are creating, as the following sections explain.

MENU Tree Objects

TITLE   Menu title for the menu bar.

ENTRY   Command string for the menu.

**********   Separator line for the menu.   Use the separator line
                          to separate commands into logical groups for the
                          end-user's convenience.

[ ]   Hollow box you can place in a drop-down menu.
                          You can use the hollow box to define the selectable

(reversible) extent of a bit image in the menu. For an example, see the Gallery Menu in the GEM Graph^{TM} application.

DIALOG Tree and PANEL Tree Objects

The same set of objects is available for DIALOG and PANEL trees. In the following descriptions, the names in parentheses are object types described in Section 6, "Object Library," in the GEM AES Reference Guide.

| BUTTON | (G_BUTTON) – Boxed string the end-user selects to indicate a choice among alternatives. |

STRING   (G_STRING) – Boxless character string, normally containing explanatory text for the end-user.

EDIT:_____   (G_FTEXT) – A formatted text field the end-user can change.

| EDIT:_____ | (G_FBOXTEXT) – A formatted text field inside a box. The end-user can change the text field. |
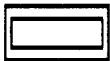
(G_IBOX) – Hollow box through which the end-user can see the fill pattern or text beneath.

(G_BOX) – Opaque (non-transparent) box.

TEXT   (G_TEXT) – Formatted text. You can select sizes, colors, fonts, and masks for these objects.

| ¢ | (G_BOXCHAR) – Single character in an opaque box. |

| BOXTEXT | (G_BOXTEXT) – Same as TEXT, but with a surrounding, opaque box. |

(G_ICON) – Use to display an icon (data plus mask) created with GEM IconEdit.

(G_IMAGE) – Use to display a bit image (data field only) created with GEM IconEdit.

ALERT Tree Objects

Alerts use an optional warning icon, up to five message lines, and up to three exit buttons. The descriptions of the icons (below) also suggest how each kind of alert might be used. This "philosophy" is entirely optional.

The default state of an exit button in an ALERT is set in your program code.

An unformatted text string telling the end-user the nature of the problem. An alert can have a maximum of five message lines, each with no more than forty characters.

NOTE icon. This is the default icon. This kind of alert can inform the end-user that he has attempted something the application does not permit. The alert might only have an OK exit button, so the user can acknowledge the message.

WAIT icon. This kind of alert can inform the end-user that he has requested a permitted action, but that the action is not possible under the present circumstances, perhaps because a drive door is open or there is no disk in the drive. The message text can suggest solutions. The alert would have two exit buttons, Cancel and Retry.

STOP icon. This kind of alert can warn the end-user that his request, while perfectly legal, can result in the loss of data. Formatting a disk is one example. The alert would have two exit buttons, OK and Cancel.

FREE Tree Objects

Free-string   An unformatted text string containing the text you wish to display.

A bit image (data field only) created with GEM IconEdit.

FREE

OBJECT CLASSES

The objects in the parts box can be divided into five classes:

- unformatted strings
- box type
- formatted text
- bit images
- icons

The following table lists the object classes, the object types in each class, and the object trees in which they can appear.

Table 3-2.   Object Classes

Object Class	Object Type	Object Tree(s)*
Unformatted String	TITLE	M
	ENTRY	M
	separator line	M
	BUTTON	D,P,A
	STRING	D,P
	Message Line	A
	Free-string	F
Box Type	all boxes, including outer box of DIALOG or PANEL	M,D,P
	single character in box	D,P
Formatted Text	TEXT	D,P
	BOXTEXT	D,P
	editable text (EDIT:____)	D,P
	boxed editable text	D,P
Bit Images	IMAGE	D,P,F
Icons	ICON	D,P

*M = MENU, D = DIALOG, P = PANEL, A = ALERT, F = FREE

Unformatted Strings and Formatted Text

GEM RCS provides two ways of entering text in object trees: unformatted strings and formatted text. The following table lists some of the characteristics of and differences between these two kinds of text.

Table 3-3.   Unformatted Strings and Formatted Text

Unformatted Strings	Attributes can be set.
	Can be aligned within parent.
	System font and color only.
	Transparent mode only (background color and pattern show through).
	Always left-aligned in extent.
Formatted Text	Attributes can be set.
	Can be aligned within parent.
	Colored text available.
	Small font available.
	Transparent mode or replace mode (background color and pattern do not show through).
	Text can be left-aligned, right-aligned, or centered within extent.
	Outline color available (boxed formatted text only).
	Outline thickness available (boxed formatted text only).

ADDING OBJECTS TO TREES

To add an object to your tree, drag its icon from the parts box to the view window. You can then Shift-drag the object in the view window to make additional copies.

The Shift-drag technique is especially useful if you want two or more objects with the same attributes. For example, if you want three buttons with attributes in common, drag one BUTTON object from the parts box, set the common attributes, and then copy it twice, rather than dragging the object from the parts box three times and setting the attributes three times.

Special Cases

MENU trees   The TITLE object can be placed only in the title bar. The ENTRY, separator line, and hollow box objects can only be placed in the menu box.

You cannot add or remove objects from the Desk Menu. The first line of the Desk Menu is the only editable entry.

ALERT trees   An ALERT tree can only contain one warning icon. To replace the default NOTE icon, drag another icon from the parts box and place it inside the alert. GEM RCS automatically deletes the NOTE icon and positions the new icon. If you don't want an icon, drag the existing one to the trash can.

An ALERT can have a maximum of three exit buttons and five message lines of no more than forty characters each. GEM RCS automatically adjusts the size of the alert box to the length of the message text.

MOVING, COPYING, AND DELETING OBJECTS

Moving and copying objects are described in Table 1-1. Also see "Using the Clipboard," later in this section.

To delete an object, do one of the following:

● Drag it to the trash can.
● Select the object and click on the trash can icon.
● Select the object and choose the **Delete** command from the Edit Menu.

SIZING OBJECTS

You can size most objects by selecting the object and then dragging its size handle.

Some objects--including STRING objects and others whose contents you edit in the Edit Unformatted String Object dialog--change size automatically when you add to the text in the object. However, such a change might violate the parent-child size relationship, which is described next.

MAINTAINING THE PARENT-CHILD SIZE RELATIONSHIP

As detailed in Section 6 of the GEM AES Reference Guide, object structure rules require that the parent object always contain its children. This means that the child cannot be larger than its parent and that the child cannot extend past the boundaries of the parent.

In several cases, GEM RCS protects against violations of the parent-child size relationship. For example, you cannot size down a menu box so that the entries or separator lines extend past the edge of the box. Similarly, you cannot enlarge a BUTTON object to extend past the edge of a dialog box.

However, by editing text strings in the Edit Unformatted String Object dialog, you can make the following objects too long to fit inside their parent objects:

● a STRING in a dialog or panel
● an ENTRY object in a menu box

For example, you can type text into an ENTRY so that the command extends past the edge of the menu box. In this case, GEM RCS displays an alert warning you that the child does not fit inside its parent. You have two options:

OK   GEM RCS accepts the long ENTRY and automatically enlarges the text string's extent to include the entire string. You should then make the menu box larger. (If you don't, be forewarned that GEM AES only writes to the menu/alert buffer and redraws the part of the screen under the menu box. Anything outside the menu box will remain on the screen as garbage.)

Cancel   GEM RCS ignores your changes and retains the original ENTRY text.

ICONS AND BIT IMAGES

Icons and bit images are both created with GEM IconEdit.

An icon consists of DATA (the "picture") and MASK, usually a solid "shadow" of the DATA. The function of the MASK is to prevent any background colors or patterns from showing through the DATA. You can edit an icon to include a text string or a single character at any of several positions relative to the icon. See the description of the Edit Icon Object Dialog, later in this section.

A bit image is DATA only. Because it has no MASK, the bit image allows any background color or pattern to show through. Unlike icons, bit images cannot be edited to include text strings or characters.

Loading Icons

To load an icon into a DIALOG or PANEL tree, drag the ICON icon from the parts box to the view window. Then select the icon and choose the **Load...** command from the Options Menu.

GEM RCS first displays a dialog asking if you want to load the icon's DATA, MASK, or both. If you load DATA only, any background color or pattern will show through the icon. If you load MASK only, you will simply block out any background color or pattern, but the icon's image will not appear.

If you choose both DATA and MASK, GEM RCS displays the Item Selector twice. The first time, select the icon's DATA. When GEM RCS immediately redisplays the Item Selector, select the icon's MASK.

Note: The files for DATA and MASK both have the extension .ICN. You'll need some naming convention to distinguish the two, like ICON<u>D</u>.ICN and ICON<u>M</u>.ICN.

Loading Bit Images

To load a bit image into a DIALOG or PANEL tree, drag the IMAGE icon from the parts box to the view window. Then select the icon and choose the **Load...** command from the Options Menu. GEM RCS displays the Item Selector, and you can select the bit image file.

Like icons, bit image files have the .ICN extension.

USING FREE TREES

As noted in Section 1, FREE trees can be used to enable context-sensitivity in your application at run-time. We'll use GEM RCS as an example of how you can use Free-string objects and FREE images.

Note: You can assemble your Free-strings and FREE images in as many FREE trees as you like, mixing and matching as the spirit moves you. However, when GEM RCS saves the resource file, it combines all Free-strings into one tree (called FRSTR1) and all FREE images into another tree (called FRIMG1).

Free-String Objects

The GEM RCS resource file contains several strings in the tree FRSTR1, including the following commands:

● **Hide Parts**
● **Show Parts**

The default form of this command (**Hide Parts**) is contained in the tree for GEM RCS's menus. When the end-user chooses the command, the application changes the pointer for the command string from the default in the MENU tree to the alternate (toggled) command in the FRSTR1 tree. The next time the GEM AES screen manager draws the menu, it uses **Show Parts** from FRSTR1. From then on, as context requires, the application swaps in and out the pointers to the two forms of the command in FRSTR1.

Note: If you change the length or spacing of the string in the tree containing the default, don't forget to make the same changes to the related free-string or strings.

FREE Image Objects

The GEM RCS clipboard (described later in this section) is a context-sensitive object. Its default appearance is a bit image object in the toolkit's object tree. The tree FRIMG1 contains its alternate appearance (dogeared, to indicate that the end-user has placed something on the clipboard) and a duplicate of the default version.

When the end-user places something on the clipboard, GEM RCS swaps in the dogeared bit image from FRIMG1. The next time the default version is needed, GEM RCS uses the duplicate from FRIMG1, and from then on GEM RCS swaps in and out the two versions from FRIMG1.

USING THE OBJECT EDITING DIALOGS

When you open an object in the view window, GEM RCS displays one of the following dialogs:

● Edit Unformatted String Object
● Edit Box Type Object
● Edit Formatted Text Object
● Edit Bit Image Object
● Edit Icon Object

The dialog displayed depends on the class of object opened (see Table 3-2).

Each object editing dialog is different (see the individual descriptions following), but in each case you can use the dialog to name or rename the object you're working on. To do so, click on the line following "Object Name" and type the name you want to assign to the object. You can then refer to this name in your program code.

Edit Unformatted String Object Dialog

The default text, BUTTON or STRING, appears in the "Text" field. To edit the field, press the Esc key to erase the text in the field and then type the new text. You can move the cursor back and forth in the field with the left- and right-arrow keys, and you can erase characters with the Backspace and Delete keys.

To help you gauge the length of your text strings, the dialog marks the "Text" field at 20, 30, and 40 characters.

See "Unformatted Strings and Formatted Text," earlier in this section, for a discussion of the differences between unformatted and formatted text.

Edit Box Type Object Dialog

If the object is the single character in a box (object type G_BOXCHAR), the "Character" field contains the default character. You can edit or erase this character as you wish. For all other box objects, the "Character" field is empty, and GEM RCS ignores any characters you enter in it.

Edit Formatted Text Object Dialog

The Edit Formatted Text Object dialog exists in two forms: one for editable text (G_FTEXT and G_FBOXTEXT objects) and another for non-editable text (G_TEXT and G_BOXTEXT objects). The ob_spec value of all four object types is a POINTER to a TEDINFO structure (see Section 6 of the GEM AES Reference Guide).

See "Unformatted Strings and Formatted Text," earlier in this section, for a discussion of the differences between unformatted and formatted text.

The dialog for non-editable formatted text has only one field, PTEXT, in which you enter text.

The dialog for editable formatted text has three fields: PTMPLT (template), PVALID (validation), and PTEXT (text entry). These fields are explained at length in Section 6 of the GEM AES Reference Guide under "TEDINFO Structure." Note that in GEM RCS, PTMPLT, PVALID, and PTEXT use a tilde (~) in place of an underline (_) to avoid

confusion between the field itself and the placeholders for editable characters.

The following example illustrates how PTMPLT, PVALID, and PTEXT create an editable text field in which the end-user can type the date.

```
PTMPLT>Today's date:  ~~/~~/~~_____
PVALID>~~~~~~~~~~~~~~99~99~99_____
  PTEXT>~~~~~~~~~~~~~~01~01~86_____
```

PTMPLT: To edit the field, press the Esc key and then type the new string as shown above. The tildes take the place of the characters the end-user can edit at run-time.

PVALID: The validation field controls the location and type of the characters the end-user enters at run-time. A "9" in PVALID indicates that the end-user may only enter a digit (0-9) in that position. Tildes in PVALID represent literal characters that must appear exactly as entered in PTMPLT.

PTEXT: The digits under the nines in the validation field appear as the default entry for the field. The end-user can type over them to change the date.

Edit Bit Image Object Dialog

This dialog contains only the "Object Name" field and a reminder to use the **Load...** command (Options Menu) to load the new bit image data. "Loading Bit Images," earlier in this section, describes how you load bit images.

Edit Icon Object Dialog

In addition to naming the object, you can use this dialog for the following:

● To enter and locate a text string you want to appear as part of the icon.

● To enter and locate a single character you want to appear as part of the icon.

The dialog contains two locator boxes: one for the Text field and

another for the Character field. You can locate a text string at the top, middle, or bottom of the icon. You can locate a character at any of nine positions relative to the icon.

See "Loading Icons," earlier in this section, for a description of how you load an icon.

USING THE CLIPBOARD

The clipboard is a storage place for trees or objects from the view window. You can move or copy items to the clipboard, but only one item can be on the clipboard at a time. Moving or copying overwrites anything currently on the clipboard.

When you place an item on the clipboard, GEM RCS dog-ears the icon to let you know there is something there.

To cut a tree or object to the clipboard, do one of the following:

● Drag its icon to the clipboard.
● Select the tree or object and then click on the clipboard icon.
● Select the tree or object and then choose the **Cut** command from the Edit Menu.

Cutting removes the tree or object from the view window.

To copy a tree or object to the clipboard, Shift-drag its icon, or select the tree or object and use the **Copy** command on the Edit Menu. Copying leaves the original icon in the view window.

To paste a tree or object from the clipboard to the view window, do either of the following:

● Drag from the clipboard icon.

● Display the Edit Menu, press the mouse button when the **Paste** command is highlighted, and drag from the menu. Use this technique if you have removed the toolkit from the screen with the **Hide Tools** command.

Either technique empties the clipboard.

To copy a tree or object back to the view window, Shift-drag the clipboard icon. The clipboard still contains the original tree or object.

Whenever you paste or copy a tree from the clipboard, GEM RCS displays the "Enter Tree Name" dialog so you can give it a unique name.

You can use the clipboard as a holding place for an object from the parts box of one tree type and then add the object to another tree. For example, you can add bit images to a MENU tree by doing the following:

1. Open a DIALOG tree.

2. Put a bit image in the DIALOG tree's view window and then cut or copy it to the clipboard.

3. Close the DIALOG tree and open a MENU tree.

4. Paste or copy the bit image from the clipboard into a menu. You might want to make the bit image the child of a hollow box. See the description of the hollow box under "MENU Tree Objects," earlier in this section.

Note: Be careful when you mix and match objects this way. Some objects and trees are incompatible, and mixing them can cause unpredictable results. See "Unknown Object Tree Types" in Section 4.

GEM RCS clears the clipboard when you choose the **New** command to begin a new resource file and when you choose the **Open...** command to edit an existing resource file. Both commands are on the File Menu.

USING THE TOOLKIT

The toolkit contains tools with which you can make a variety of changes to a selected object, include the following:

● changing the object's color
● changing the object's fill pattern
● aligning the object within its parent
● setting certain attributes for the object
● changing the object's line thickness

A tool is disabled if its effect is not meaningful for the selected object. For example, the fill pattern tool is disabled for STRING objects because STRING objects cannot have a fill pattern.

When you move the pointer over the toolkit, GEM RCS highlights only the tools enabled for the currently selected object. To use a tool, click on the highlighted icon. GEM RCS displays a pop-up menu of the tool's options.

To choose from a pop-up menu, drag through the menu and click when the option or command you want is highlighted. If you click outside the menu, it disappears, and the object is unaffected.

Descriptions of the Tools

Selects a background color for an object. The menu appears in color if your computer can display color. Otherwise, assign colors using the numbered color codes. These correspond to the codes described under "Object Colors" in Section 6 of the <u>GEM AES Reference Guide</u>.

Selects a fill pattern for an object.

Note: For a fill pattern to be visible, it must also have a visible color. The defaults are white color and transparent fill pattern.

Selects a color for a TEXT, BOXTEXT, FTEXT, or FBOXTEXT object. The menu appears in color if your computer can display color.

Aligns text, changes text size, or changes the background. For TEXT, BOXTEXT, FTEXT, and FBOXTEXT objects only. Most of the commands are self-explanatory; explanations and an illustration of **Transparent** and **Replace** follow.
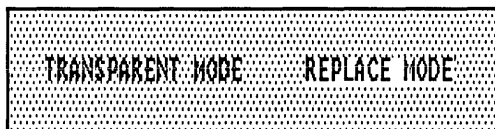
Transparent – Makes each character cell (the text's immediate background) transparent; the color and fill pattern of the box containing the text shows through.

Replace – Makes each character cell opaque; the text appears on a solid white background.
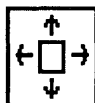
Aligns or resizes an object within the boundaries of its parent. The alignment options are self-explanatory; explanations of the fill and snap options follow.

Fill Horizontal – Automatically resizes the object so that it extends from the extreme left to the extreme right boundaries of the parent object.

Fill Vertical – Automatically resizes the object so that it extends from the top to the bottom boundaries of the parent object.

Character Snap – Snaps the selected object in a PANEL tree to the nearest character boundary. This is useful for precise alignment of objects in the PANEL, which otherwise allows free placement of objects.

Selects attributes for an object. GEM RCS places a marker next to any attribute assigned to the current object. The attributes in the upper half of the menu set object flags (ob_flag) and affect the object's interaction with the FORM_DO call. The attributes in the lower half of the menu set object states (ob_state) and affect how the Object Library draws the object. See Section 6, "Object Library," and Section 7, "Form Library," in the GEM AES Reference Guide.

Explanations of the attributes follow.

Selectable – The end-user can select the object by clicking on it. Clicking displays the object in reverse video.

Exit – Clicking on the object fulfills an exit condition and causes FORM_DO to finish processing and return a value. Typically assigned to the exit buttons in a dialog. Objects with the Exit attribute <u>must</u> also have the Selectable attribute. They can also have the Default attribute, but that is optional.

Default – The object is selected automatically when the end-user presses the Enter key at run-time. If you designate more than one object in a tree as the default, you won't receive an error, but your results are not predictable.

Note: The Exit and Default attributes each add a pixel to all sides of the border of a BUTTON object. Thus, a button with only the Exit attribute has a two-pixel border, and a button with both attributes has a three-pixel border.

Radio Button – Makes the object a member of a set of "radio buttons." Radio buttons are like the buttons on a car radio; pressing one button makes another one pop out, which means the end-user can only select one radio button at a time. Members of a set of radio buttons must be nested at the same level within a common parent object.

Touchexit – Pressing the mouse button while the pointer is on the object fulfills an exit condition and causes FORM_DO to finish processing and return a value. The application does not wait for the mouse button to come up. Objects with the Touchexit attribute must <u>not</u> have the Selectable attribute.

Editable – The end-user can change information in the object at run-time. Use only with editable text objects.

Hidden – Hides the object so that it is not displayed on the screen. Use the **Unhide Children** command on the Hierarchy Menu to display hidden objects.

Shadowed – Draws a drop shadow around the object (usually a box). Shadows and outlines (see below) conflict.

Checked – Draws a triangle inside the left margin of the object.

Outline – Draws an outline around a boxed object. Outlines conflict with shadows and have no effect on boxes with outward borders.

Crossed – Draws an X through the object in the system background color. Use only with boxed objects.

Disabled – Draws the object at half-intensity (gray). The end-user cannot select a disabled object.

Selected – The object is preselected (appears in reverse video) when the end-user sees the tree.
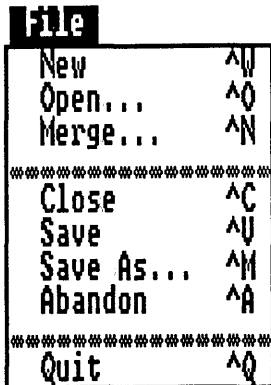
Selects a color for the border around a boxed object. The menu appears in color if your computer can display color.

Selects a border thickness for any boxed object but a BUTTON. The default thickness is a single pixel around the outside of the object (the bottom of the left column in the menu). The left column also provides two- and three-pixel thicknesses outside the object. The right column provides one-, two, and three-pixel thicknesses inside the object. The option in the lower center of the menu is the null (invisible) border. The fill patterns in the pop-up menu are for illustration only and do not affect the fill pattern in the object. See the illustration in Section 2.

GEM RCS MENU COMMANDS

Note: Commands are disabled (dimmed) when choosing them is not meaningful in the present context of GEM RCS. For example, the Edit Menu commands (**Cut**, **Copy**, **Paste**, and **Delete**) are disabled when no tree or object is selected.

```
 File 
┌─────────────┐
│New       ^W │
│Open...    ^O│
│Merge...   ^N│
│░░░░░░░░░░░░░│
│Close      ^C│
│Save       ^U│
│Save As... ^M│
│Abandon    ^A│
│░░░░░░░░░░░░░│
│Quit       ^Q│
└─────────────┘
```

New	Clears the view window so you can start a new resource file.
Open...	At the root level only, and with no tree icon selected, displays the Item Selector so you can open an existing resource file.
	If you have a tree or object currently selected in the view window, this command opens that tree or object.
Merge...	Displays the Item Selector so you can select an existing resource file to merge with the one on which you are working.
	Merging resource files can produce name conflicts. If the merged file contains names already used in the current file, GEM RCS creates new names for the duplicates. (Printing the .H or .I file is a good way to check for duplicates--see the **Output...** command in the Global Menu.)

Close   Closes the current resource file or tree. Closing a file clears it from the screen. (If you have not saved your current edits, GEM RCS displays a dialog asking whether to save or abandon them. See the note below.) Closing a tree returns you to the root level of the file. Clicking on the close box has the same effect as choosing this command.

Save   Saves your current resource file. The file remains in your workspace so you can continue editing. Use this command periodically so you won't lose all of your work to a power failure or computer malfunction.

Save As...   Saves your current resource file under a name you provide. Use this command in either of the following situations:

- To name and save a resource file for the first time.

- To save an edited version of an existing file under a new name and/or to a different directory. The new directory path and/or filename appear in the title bar. The original version of the file remains on disk under the old name.

Abandon   Abandons all edits on the current file since the last time it was saved.

Quit   Stops GEM RCS and returns you to the GEM Desktop.

Note: If you edit a new or existing resource file and then choose the **New, Open..., Close, Abandon,** or **Quit** command without first saving your edits, GEM RCS displays a dialog that asks if you want to abandon your edits. The dialog's three exit buttons offer you these options:

- Abandon the edits.
- Save the file.
- Cancel the requested command.

```
┌──────────┐
│ Global   │
├──────────────────────┐
│ Output...        ⬥O  │
│ Protection...    ⬥S  │
│ Save Preferences ⬥R  │
│ ▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨▨ │
│ Hide Parts       ⬥P  │
│ Hide Tools       ⬥H  │
└──────────────────────┘
```

Output...   Selects the types of output files (in addition to the .RSC and .DFN files) GEM RCS creates when you save resource files. You can select include files for C (the default), Pascal, BASIC, and FORTRAN-77, as well as a .RSH source file for use with RSCREATE (see Section 4).

Protection...   Sets the level of protection GEM RCS applies while you are editing resource files. This command has the following options:

* LOCKED – Permits editing and sizing but forbids changes to the tree structure. This setting is intended for post-production changes, such as translating strings for internationalization of your program. Values associated with tree and object names are preserved; you need not recompile the application after editing.

* NORMAL – The default protection level. It warns you before rearranging trees. NORMAL permits all operations but warns you of impending changes in parent-child relationships and workspace clearance.

* EXPERT – Gives you no warnings. You can make whatever changes you wish to the resource file.

Save Preferences Makes the settings and selections in the output and protection dialogs the defaults for subsequent GEM RCS sessions; saves them to a file called RCS.INF in the GEMAPPS folder. To restore the original defaults, delete RCS.INF.

Hide Parts   Hides the parts box on your screen. The command toggles between **Hide Parts** and **Show Parts**.

Hide Tools   Hides the toolkit on your screen. The command toggles between **Hide Tools** and **Show Tools**.

```
Edit
 Cut      ♦C
 Copy     ♦Y
 Paste    ♦A
~~~~~~~~~~~~~~
 Delete   ♦D
```
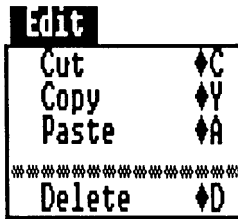
Cut	Cuts the currently selected tree or object to the clipboard. The original no longer appears in view window.
Copy	Copies the currently selected tree or object to the clipboard. The original remains in view window.
Paste	Lets you move a tree or object from the clipboard to the view window by dragging from the Edit Menu. (Use when toolkit is not visible.)
Delete	Deletes the selected tree, object, or group of objects.

```
┌─────────────┐
│ Options     │
│  Info...   ⬧I │
│  Name...   ⬧N │
│  Type...   ⬧T │
│  Load...   ⬧L │
└─────────────┘
```

Info...   Shows you information about the current file or the
                     currently selected tree or object. Also shows how
                     much of your workspace you have used and how
                     much you still have available.

Name...   Displays a dialog you can use to name or rename
                     an object tree.

Type...   Displays a dialog you can use to change a tree or
                     object's type.

Load...   Displays the Item Selector so you can replace an
                     IMAGE or ICON object with one created using GEM
                     IconEdit.

Hierarchy
Sort Children...   ♦F
Unhide Children   ♦U
※※※※※※※※※※※※※※※※※※※※※※※※
Remove Parent

Sort Children...   Displays a dialog in which you can specify how objects should be sorted within their parent. Sorting causes the objects in the tree to be drawn in a logical order. Without sorting, the objects in the tree are drawn in the order in which they were added to it. This has no effect on the functionality of the tree, but it can be visually displeasing,

Unhide Children   Causes any hidden children of the currently selected object to be displayed.

Remove Parent   Removes the currently selected object without removing its children. (Dragging an object to the trash also removes its children.)

Note: Resorting children after your application has been compiled can renumber the objects in your tree, causing any references to them in the program code to be incorrect. (You can check this by looking at the .H file created by GEM RCS.) If incorrect references exist, the program code must be recompiled.

```
        ████RCS█████
 ┌─────────────────────────┐
 │ About GEM RCS...        │
 │▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓│
 │  Desk Accessory         │
 │  Desk Accessory         │
 │  Desk Accessory         │
 │  Desk Accessory         │
 │  Desk Accessory         │
 │  Desk Accessory         │
 └─────────────────────────┘
```
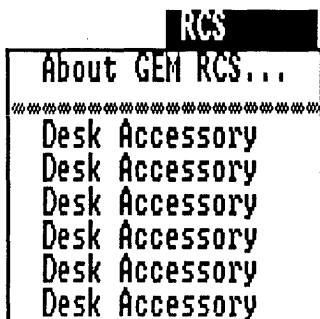
About GEM RCS... Displays a dialog showing information about GEM RCS, including the version number, copyright, and authors' names.

Desk accessories Names of desk accessories currently available on your system. The menu can display up to six desk accessory names.

KEYSTROKE EQUIVALENTS OF MENU COMMANDS

Most of the GEM RCS commands can be executed by clicking on the command in the menu or by typing a keystroke combination. The following table lists the commands in the order they appear in the menus and gives their keystroke equivalents.

Note: A command's keystroke equivalent is only enabled if the command itself is enabled.

Table 3-4.   Keystroke Equivalents of Menu Commands

Command	Keystroke Equivalent	Menu
New	Ctrl-W	File
Open...	Ctrl-O	File
Merge...	Ctrl-N	File
Close	Ctrl-C	File
Save	Ctrl-V	File
Save as...	Ctrl-M	File
Abandon	Ctrl-A	File
Quit	Ctrl-Q	File
Output...	Alt-O	Global
Protection...	Alt-S	Global
Save Preferences	Alt-R	Global
Hide/Show Parts	Alt-P	Global
Hide/Show Tools	Alt-H	Global
Cut	Alt-C	Edit
Copy	Alt-Y	Edit
Paste	Alt-A	Edit
Delete	Alt-D	Edit
Info...	Alt-I	Options
Name...	Alt-N	Options
Type...	Alt-T	Options
Load...	Alt-L	Options
Sort Children...	Alt-F	Hierarchy
Unhide Children	Alt-U	Hierarchy

End of Section 3

RSCREATE and Other Technical Information

USING RSCREATE

RSCREATE is a C language utility program that creates a resource file. It is primarily intended for two purposes:

● To create a resource file from a hand-edited file.
● To port resource files between microprocessor environments.

RSCREATE uses a .RSH file (described next) as an include file. To generate a new .RSC file with RSCREATE, include the .RSH file in RSCREATE.C and compile, link, and run RSCREATE.

.RSH File

The .RSH file, which is an ASCII file you can edit with a text editor or word processor, is one of the optional GEM RCS output files. To make GEM RCS create a .RSH file, choose the **Output...** command from the Global Menu, and click on the box labeled "*.RSH" in the output file dialog. You can make the .RSH file one of the default set by choosing the **Save Preferences** command from the Global Menu. See the description of the Global Menu in Section 3.

GEM RCS only produces output files as part of the process of saving a resource file. For that reason, if you open a resource file solely to produce a .RSH file (in other words, without editing the file), you must use the **Save As...** command and then enter the file's current name in the Item Selector. Because the resource file has not been edited, closing the file with the **Close** command or the close box does not save the file and does not create a .RSH file.

Hand-Edited Resource Files

To hand-edit a resource file, generate a .RSH file and make your changes in the .RSH file with a word processor or text editor. Before you hand-edit a resource file, however, be sure you are familiar with the material in the GEM AES Reference Guide describing the Object Library and the Form Library.

When you hand-edit a .RSH file, remember the following:

● When adding a new object, TEDINFO, etc., insert it at the end of the current entries.

● If you insert new objects into existing trees, update the tree base definitions.

● GEM RCS expects object trees in the following order: root first, followed by its children, left to right, with the rule applied recursively. Make sure you enter any hand-built trees in this same order.

● If you are creating a resource file with GEM RCS and you expect to hand-edit the file later, don't enter any object names while you're in GEM RCS. Wait until you've edited the .RSH file and have run it through RSCREATE. Then read the new .RSC file back into GEM RCS and enter the object names.

   If you enter object names in your first pass through GEM RCS, the .H and .DFN files are linked to the object numbers assigned in that first pass. RSCREATE changes these object numbers, and GEM RCS will not be able to use the .DFN file for any subsequent work on the resource file.

● GEM RCS automatically sets the LASTOB flag in the OBJECT structure of the last object in each tree. (See the descriptions of the OBJECT structure and object flags in Section 6 of the GEM AES Reference Guide.) If you hand-edit a tree, make sure this flag is set for the last object in the tree. If you don't, your application can crash at run-time.

Porting Resource Files

To port a resource file from one environment to another, do the following:

1. Move the .RSH file to the new environment.

2. Add a header to make the .RSH file compatible with the target environment. If you are porting to a 68K environment, the header is commented out in RSCREATE.

3. Enter the target-format .RSH as an include file in RSCREATE.

4. Compile, link, and execute RSCREATE on the target environment.

Compiler Notes

Compile RSCREATE with Lattice™ C or another full language implementation.

Some C compilers (including Lattice C) fold duplicate strings together when you compile RSCREATE. This can present problems at run-time if the strings are to be edited. In that case, run the resource file through GEM RCS again to resolve the duplicate strings.

CHAINING RESOURCES

The GEM AES enforces an absolute limit of 64K for a resource file. If you need a larger resource, you can chain two resource files together. However, to use the second file in the chain, you must first clear from the AES any references to the first resource file, and you must clear the first file from memory by making a RSRC_FREE call before you load the second file.

UNKNOWN OBJECT TREE TYPES

If you load a resource file and do not have its .DFN file, GEM RCS displays each tree as an UNKNOWN (represented by a question mark) or an ALERT. Before you can work on the file, you must convert the unknown trees to one of the known types. You should also give each tree a name for the .DFN file you will create when you save the file.

To assign types and names to the trees, do the following:

1. Click on the first UNKNOWN (TREE1) to select it.

2. Display the Options Menu, choose the **Type...** command, and select the DIALOG option. This replaces the question mark icon with the DIALOG icon.

3. Double-click on the TREE1 icon. This opens the tree, and you can then see what it is.

If the tree really is a dialog, do the following:

1. Close the tree to return to the root level.
2. Choose the **Name...** command from the Options Menu.
3. Give the tree a name.

If the tree is not a dialog, do the following:

1. Close the tree to return to the root level.
2. Choose the **Type...** command from the Options Menu.
3. Select the correct type for the tree.
4. Choose the **Name...** command from the Options Menu.
5. Give the tree a name.

For the ALERT trees, you need only open them to see what their messages are and then use the **Name...** command to give them names.

Using the **Type...** command from the Options Menu, you can change the type designation of any tree or object. Remember that MENU and ALERT trees are quite restrictive; they accept only specific objects that are for the most part not compatible with other object trees. You can change a tree's type from a restrictive type to a less restrictive type (for example, from ALERT to DIALOG), but you should not try to go from less restrictive to more restrictive. To do so can produce unpredictable results.

ARRANGING AND ALIGNING OBJECTS

Especially in a dialog, you might want to have several objects arranged in a particular way. For example, you might want three or four text strings to appear in a right-aligned column. To do this, you can first arrange them inside a box, as follows:

1. Drag a box from the parts box to the view window. Make the box the size you want.

2. Add the STRING objects to the box and edit them.

3. Select all of the strings in the box using the Shift-click technique.

4. Display the alignment pop-up menu and choose the alignment you want. (As long as the strings are inside their parent box, you can also move them as a group.)

5. Select the parent box and then choose the **Remove Parent** command from the Hierarchy Menu. Removing the parent box reduces the size of both the tree and the resource file.

CREATING A LIBRARY FILE

If you use certain icons, bit images, or subtrees frequently, you can collect them in a "library" resource file. Using the **Merge...** command (File Menu), you can read this library file into your current resource file. You can then use the trash can or clipboard to cut any extraneous objects from the resource file.

End of Section 4

Index

G

GEM AES, 4-3
GEM Graph, 3-5
GEM IconEdit, 3-28
GEM RCS
  levels, 1-5
  memory use, 1-9
  quitting, 2-15
  starting, 2-1
  workspace, 1-9
Global Menu, 3-25

H

Hidden attribute, 3-21
Hide Parts command, 3-26
Hide Tools command, 3-26
Hierarchy Menu, 2-13, 3-29

I

Icon objects, 3-16
  aspect ratio differences, 3-1
  loading, 3-12
Include files, 2-15, 3-25
Info... command, 1-9, 3-28
Item Selector, 2-14

K

Keystroke equivalents, 2-5

L

LASTOB flag, 4-2
Levels
  object level, 1-5
  root level, 1-5
Library file, 4-5
Load... command, 3-12, 3-13,
    3-16, 3-28
Loading bit images, 3-13
Loading icons, 3-12
LOCKED protection setting, 3-25

M

Menu bar, 1-5
Menu box
  sizing, 2-4, 2-6
MENU trees, 1-2, 3-4, 3-10, 4-4
Menu/alert buffer, 1-3
Menus
  adding ENTRY, 2-4
  adding TITLE, 2-3
  command conventions, 2-5
  copying an ENTRY, 2-4
  creating, 2-1
  keystroke equivalents, 2-5
  sizing command extent, 2-6
  sizing menu box, 2-4, 2-6
  sorting, 2-7
  symbols for Ctrl and Alt keys,
    2-5
Merge... command, 3-23, 4-5
Message line (ALERT trees), 3-6
Mouse techniques, 1-7

N

Name... command, 3-28, 4-4
Naming objects, 2-9
New command, 3-23, 4-8
NORMAL protection setting,
    3-25
NOTE icon, 3-6
  replacing, 3-10

O

Ob_spec value, 3-15
Object level, 1-5
Object names, 2-9, 4-2
Object trees
        See "Trees"
Object types
  G_BOX, 3-5
  G_BOXCHAR, 3-5, 3-15
  G_BOXTEXT, 3-5, 3-15
  G_BUTTON, 3-5
  G_FBOXTEXT, 3-5, 3-15
  G_FTEXT, 3-5, 3-15
  G_IBOX, 3-5
  G_ICON, 3-6
  G_IMAGE, 3-6
  G_STRING, 3-5
  G_TEXT, 3-5, 3-15
Objects, 1-2
  adding to dialog, 2-8
  adding to trees, 2-8, 3-9
  aligning, 3-20
  arranging/aligning, 4-4
  background color, 3-19
  border color, 3-22

border thickness, 3-22
box for user-defined objects
        (MENU trees), 2-3, 3-4
changing type, 4-4
children, 1-2
classes, 3-7
closing, 1-6
copying, 1-7, 2-4, 2-9, 3-10
copying from clipboard, 3-17
copying to clipboard, 3-17
deleting, 1-8, 3-11
drawing order, 2-13
editing text in, 2-9
fill pattern, 3-19
icons in parts box, 3-4
in ALERT trees, 3-6
in DIALOG trees, 3-5
in FREE trees, 3-7
in MENU trees, 3-4
in PANEL trees, 3-5
LASTOB flag, 4-2
levels, 2-13
library file, 4-5
moving, 1-7
moving as a group, 4-4
moving from clipboard, 3-17
moving to clipboard, 3-17
naming, 2-9, 3-14
opening, 1-7
order in trees, 4-2
parent-child size relationship,
        3-11
parents, 1-2
radio buttons, 2-10
selecting, 1-7
sizing, 1-8, 2-9, 3-11, 3-20
sorting, 2-7, 2-13, 3-29

KERMIT User's Guide

for GEM^{TM} Programming

Adapted from
Fifth Edition, Revision 1

Frank da Cruz, editor

Columbia University Center for Computing Activities
New York, New York 10027

Permission for use of KERMIT (file transfer protocol) has been granted by Columbia University Center for Computer Activities. Cost of inclusion of KERMIT in this product is nominal. KERMIT is available from Columbia University for many systems.

No warranty of the software nor of the accuracy of the documentation surrounding it is expressed or implied, and neither the authors nor Columbia University acknowledge any liability resulting from program or documentation errors.

For information on how to obtain KERMIT, contact:

KERMIT Distribution
Columbia University Center for Computing Activities
612 West 115th Street
New York NY 10025

The KERMIT protocol was named after Kermit the Frog, star of the television series The Muppet Show, and is used by permission of Henson Associates, Inc.

TRADEMARKS

Contents

Contents

4 When Things Go Wrong

5 Sample KERMIT Sessions

Tables

How to Use KERMIT

KERMIT FILE TRANSFER PROTOCOL

KERMIT is a protocol for transferring files between computers of all sizes over ordinary asynchronous telecommunication lines using packets, checksums, and retransmission to promote data integrity. KERMIT is non-proprietary, thoroughly documented, well tested, and in wide use. The protocol and the original implementations were developed at Columbia University and have been shared with many other institutions, some of which have made contributions of their own.

KERMIT IMPLEMENTATION

This implementation of KERMIT is specifically intended to enable the following modes of communication:

- PC DOS systems to GEM^{TM} DOS systems
- GEM DOS systems to PC DOS systems
- PC DOS systems to PC DOS systems
- GEM DOS systems to GEM DOS systems

As used in this guide, "PC DOS" refers generically to PC DOS and MS^{TM}-DOS. "GEM DOS" refers both to GEM DOS and operating systems based on GEM DOS.

BASIC KERMIT COMMANDS

These are generic descriptions of the most basic KERMIT commands. Detailed descriptions will come later. In these descriptions, local refers to the system that you are using directly, remote refers to the system to which you are CONNECTed via KERMIT. Commands may take one or more operands on the same line, and are terminated by a carriage return.

SEND filespec   Send the file or file group specified by filespec from this KERMIT to the other. The name of each file is passed to the other KERMIT in a special control packet, so it can be stored there with the same name. A file group is usually specified by including "wildcard" characters like "*" in the file specification. Examples:

```
send foo.txt
send *.for
```

CONNECT   Make a "virtual terminal" connection to the remote system. On the PC this means sending all keyboard input out the serial port, and displaying all input from the serial port on the screen. To "escape" from a virtual terminal connection, type KERMIT's escape character (CTRL-], control-rightbracket), followed by the letter "C" for "Close Connection".

SET   Establish various nonstandard settings, such as communication line number, parity, or flow control.

SHOW   Display the values of SET options.

HELP   Type a summary of KERMIT commands and what they do.

BYE   Exit local KERMIT and cause remote KERMIT to leave server mode.

EXIT   Exit from KERMIT back to the host operating system.

?   Typed anywhere within a KERMIT command: List the commands, options, or operands that are possible at this point. This command may or may not require a carriage return, depending on the host operating system.

KERMIT SERVER

A KERMIT server is a KERMIT program that does not interact directly with the user, but only with another KERMIT program. You do not type commands to a KERMIT server, you merely start it at one end of

the connection, and then type all further commands at the other end. The server is run on the remote computer, which is normally the microcomputer receiving the files.

You can give as many SEND and GET commands as you like, and when you're finished transferring files, you can give the BYE command, which sends a message to the remote KERMIT server to log itself out.

Here's an example of the use of a KERMIT server. The user is sitting at a PC DOS computer, and a GEM DOS computer's KERMIT is in server mode.

```
C>KERMIT
  IBM-PC Kermit-MS V2.26
  Type ? for help

Kermit-MS>GET SAMPLE.GEM
Kermit-MS>SEND *.APP
Kermit-MS>BYE
```

Here are basic the commands available for talking to servers.

SEND filespec   Sends a file or file group from the local host to the remote host in the normal way.

GET filespec   Ask the remote host to send a file or file group. Example:

```
get *.c
```

BYE   Shut down the remote server and exit from KERMIT.

End of Section 1

PC DOS KERMIT Commands

COMMAND INTERFACE

KERMIT has an interactive keyword-style command interface, which is roughly as follows: In response to the "KERMIT-MS>" <u>prompt</u> you may type a keyword, such as SEND, SERVER, or BYE, possibly followed by additional keywords or operands, each of which is called a <u>field</u>. You can abbreviate keywords (but not file names) to any length that makes them distinguishable from any other keyword valid for that field. You can type a question mark at any time to get information about what's expected or valid at that point.

In this example, the user types "set" and then a question mark to find out what the SET options are. The user then continues the command at the point where the question mark was typed, adding a "d" and another question mark to see what set options start with "d". The user then adds a "u" to select "duplex" (the only SET option that starts with "du") followed by an ESC (shown here by a dollar sign) to complete the current field and issue the guide word "(to)" for the next one, then another question mark to see what the possibilities are, and so forth. The command is finally terminated by a carriage return. Before carriage return is typed, however, the command can be edited using the Backspace key. Finally, the same command is entered again with a minimum of keystrokes, with each field abbreviated to its shortest unique length. In the example, the parts the user types are in boldface type; all the rest is system typeout:

```
Kermit-MS>set ? one of the following:
 debugging    delay        duplex        escape
 file         handshake    IBM           line
 parity       receive      send
Kermit-MS>set d? one of the following:
 debugging    delay        duplex
Kermit-MS>set du$plex (to) ? one of the following:
 full    half
Kermit-MS>set duplex (to) h$alf
Kermit-MS>set du h
```

NOTATION

The command descriptions use the following notation:

anything   A parameter – the symbol in italics is replaced by an argument of the specified type (number, filename, etc).

[*anything*]   Optional field. If omitted, defaults to an appropriate value.

number   A whole number, entered in prevailing notation of the system.

character   A single character, entered literally, or as a number (perhaps octal or hexadecimal) representing the ASCII value of the character.

floating-point-number

      A "real" number, possibly containing a decimal point and a fractional part.

filespec   A file specification: a filename and extension. Can also include a search path, device/directory name or other qualifying information, and "wildcard" or pattern-matching characters to denote a group of files.

^x   A control character may be written using "uparrow" or "caret" notation. Control characters are produced by holding down the key marked Ctrl or Control and typing the appropriate character--for example, Ctrl-X.

Commands are shown in upper case, but can be entered in any combination of upper and lower case.

SUMMARY OF KERMIT COMMANDS

Here is a brief list of the KERMIT commands described in this manual.

For exchanging files:
    SEND, GET

For connecting to a remote host:
    CONNECT, SET PORT, SET PARITY, SET BAUD

For acting as a server:
    SERVER

For talking to a server:
    BYE, GET, SEND

For interrupting transmission:
    Control-X, Control-Z, Control-C, Control-E

Getting information:
    HELP, SHOW

Leaving the program:
    BYE, EXIT

If you have a file called KERMIT.INI in your default or home disk, KERMIT will execute the commands in it upon initial startup. KERMIT.INI may contain any KERMIT commands (for instance, SET commands) to configure KERMIT to various systems or communications media. KERMIT.INI must either be in the current connected directory or in a directory in the search path.

SEND

Syntax:

Sending a single file:

    SEND NONWILD-FILESPEC1 [NONWILD-FILESPEC2]

Sending multiple files:

    SEND WILD-FILESPEC1

The SEND command causes a file or file group to be sent to the other system. There are two forms of the command, depending on whether filespec1 contains "wildcard" characters. Use of wildcard characters is the method of indicating a group of files in a single file specification. For instance, if FOO.FOR is a single file, a FORTRAN program named FOO, then *.FOR might be a group of FORTRAN programs.

Sending a File Group

If filespec1 contains wildcard characters, then all matching files will be sent, in directory-listing order.

Sending a Single File

If filespec1 does not contain any wildcard characters, then the single file specified by filespec1 will be sent. Optionally, filespec2 may be used to specify the name under which the file will arrive at the target system; filespec2 is not parsed or validated locally in any way. If filespec2 is not specified, the file will be sent with its own name.

SEND Command General Operation

Files will be sent with their filename and filetype (for instance FOO.BAR).

The sending KERMIT will also ask the other KERMIT whether it can handle a special prefix encoding for repeated characters. If it can, then files with long strings of repeated characters will be transmitted very efficiently. Columnar data, highly indented text, and binary files are the major beneficiaries of this technique.

If you're running KERMIT locally, you should have already run KERMIT on the remote system.

Once you give KERMIT the SEND command, the name of each file will be printed on your screen as the transfer begins, and information will be displayed to indicate the packet traffic. When the specified operation is complete, the program will sound a beep, and the status of the operation will be indicated by a message like OK, Complete, Interrupted, or Failed.

If you see many packet retry indications, you are probably suffering from a noisy connection.

If you notice a file being sent that you do not really want to send, you may cancel the operation immediately by typing either Control-X or Control-Z. If you are sending a file group, Control-X causes the current file to be skipped and KERMIT to go on to the next file, while Control-Z cancels sending the entire group and returns you to KERMIT command level.

GET

LOCAL ONLY -- Syntax: GET [REMOTE-FILESPEC]

The GET command requests a remote KERMIT server to send the file or file group specified by remote-filespec.

The GET command can be used only when KERMIT is local, with a KERMIT server on the other end of the line.

The remote filespec is any string that can be a legal file specification for the remote system; it is not parsed or validated locally. As files arrive, their names will be displayed on your screen, along with a continuous indication of the packet traffic. As in the SEND command, you may type Control-X to request that the current incoming file be canceled, Control-Z to request that the entire incoming batch be canceled.

If the remote KERMIT is not capable of expanding wildcards, then you will probably get an error message back from it like "file not found". (Remember that the Atari^{TM} ST does not support the use of wildcard characters with the GET command.)

Optional Syntax: If you are requesting a single file, you may type the GET command without a filespec. In that case, KERMIT programs that

implement the optional GET syntax will prompt you for the remote filespec on the subsequent line, and the name to store it under when it arrives on the line after that:

```
Kermit-MS>get
  Remote Source File: aux.txt
  Local Destination File: auxfile.txt
```

SERVER

REMOTE ONLY -- Syntax: SERVER

The SERVER command instructs KERMIT to cease taking commands from the keyboard and to receive all further instructions in the form of KERMIT packets from another system.

After issuing this command, go back to your local system and issue SEND, GET, BYE, or other server-oriented commands from there. When you are finished transfering files, use the BYE command to shut down and log out the KERMIT server when you are done with it.

Any nonstandard parameters should be selected with SET commands before putting KERMIT in server mode.

BYE

LOCAL ONLY -- Syntax: BYE

When running as a local KERMIT talking to a KERMIT server, use the BYE command to shut down and log out the server. This will also exit from the local KERMIT.

EXIT

REMOTE ONLY -- Syntax: EXIT

When running as a KERMIT server, use Ctrl-C and then the EXIT command to return to the host operating system after the file transfer is completed.

CONNECT

LOCAL ONLY -- Syntax: CONNECT [TERMINAL-DESIGNATOR]

Establish a terminal connection to the system at the other end of the communication line, usually by means of the the serial port. Get back to the local KERMIT by typing the escape character followed by a single character "argument." Several single-character arguments are possible:

   M   Toggle status line
   C   Close the connection and return to the local KERMIT.
   S   Show status of the connection.
   B   Send a BREAK signal.
   Q   Quit logging session transcript.
   R   Resume logging session transcript.
   ?   List all the possible single-character arguments.
   ^]   Typing the escape character twice sends one copy of it to the connected host.

SET

Syntax: SET PARAMETER [OPTION] [VALUE]

Establish or modify various parameters for file transfer or terminal connection.

You can use the SET ESCAPE command to define a different escape character, and SET PARITY, SET DUPLEX, SET FLOW-CONTROL, SET HANDSHAKE to establish or change those parameters.

When a file transfer operation begins, the two KERMITs automatically exchange special initialization messages, in which each program provides the other with certain information about itself. This information includes the maximum packet size it wants to receive, the timeout interval it wants the other KERMIT to use, the number and type of padding characters it needs, the end-of-line character it needs to terminate each packet (if any), the block check type, the desired prefixes for control characters, characters with the "high bit" set, and repeated characters. Each KERMIT program has its own preset

"default" values for these parameters, and you normally need not concern yourself with them. You can examine their values with the SHOW command; the SET command is provided to allow you to change them in order to adapt to unusual conditions.

The following parameters may be SET:

BAUD-RATE	Set the speed of the current communications port
DUPLEX	For terminal connection, full (remote echo) or half (local echo)
ESCAPE	Character for terminal connection
FLOW-CONTROL	Selecting flow control method, like XON/XOFF
HANDSHAKE	For turning around half duplex communication line
INCOMPLETE	What to do with an incomplete file
PARITY	Character parity to use
PORT	For switching communication ports
PROMPT	For changing the program's command prompt

SET BAUD-RATE

Set or change the baud rate (approximate translation: transmission speed in bits per second) on the currently selected communications device. The way of specifying the baud rate varies from system to system; in most cases, the actual number (such as 1200 or 9600) is typed. Systems that do not provide this command generally expect that the speed of the line has already been set appropriately outside of KERMIT.

SET DUPLEX

Syntax: SET DUPLEX KEYWORD

For use when CONNECTed to a remote system. The keyword choices are FULL and HALF. FULL means the remote system echoes the characters you type, HALF means the local system echoes them. FULL is the default, and is used by most hosts. Half duplex is also called "local echo."

SET ESCAPE

Syntax: SET ESCAPE <u>CHARACTER</u>

Specify or change the character you want to use to "escape" from remote connections back to KERMIT. This would normally be a character you don't expect to be using on the remote system, perhaps a control character like ^\, ^], ^^, or ^_. Most versions of KERMIT use one of these by default. After you type the escape character, you must follow it by a single-character "argument", such as "C" for Close Connection. The arguments are listed under the description of the CONNECT command, earlier in this section.

SET FLOW-CONTROL

Syntax: SET FLOW-CONTROL <u>OPTION</u>

For communicating with full duplex systems. System-level flow control is not necessary to the KERMIT protocol, but it can help to use it if the same method is available on both systems. The most common type of flow control on full duplex systems is XON/XOFF.

SET HANDSHAKE

Syntax: SET HANDSHAKE <u>OPTION</u>

For communicating with half duplex systems. This lets you specify the line turnaround character sent by the half duplex host to indicate it has ended its transmission and is granting you permission to transmit. When a handshake is set, KERMIT will not send a packet until the half duplex host has sent the specified character (or a timeout has occurred). The options may include:

NONE	No handshake; undo the effect of any previous SET HANDSHAKE.
XOFF	Control-S.
XON	Control-Q.
BELL	Control-G.
CR	Carriage Return, Control-M.
LF	Linefeed, Control-J.
ESC	Escape, Control-[.

SET INCOMPLETE

Syntax: SET INCOMPLETE OPTION

Specify what to do when a file transfer fails before it is completed. The options are DISCARD (the default) and KEEP. If you choose KEEP, then if a transfer fails to complete successfully, you will be able to keep the incomplete part that was received.

SET PORT

Syntax: SET PORT TERMINAL-DESIGNATOR

Specify the communications port for file transfer or CONNECT. This command is found on microcomputer KERMITs that run in "local" mode. SET PORT does not change the remote/local status but simply selects a different port for local operation.

SET PARITY

Syntax: SET PARITY KEYWORD

Parity is a technique used by communications equipment for detecting errors on a per-character basis; the "8th bit" of each character acts as a check bit for the other seven bits. KERMIT uses block checks to detect errors on a per-packet basis, and it does not use character parity. However, some systems that KERMIT runs on, or equipment through which these systems communicate, may be using character parity. If KERMIT does not know about this, arriving data will have been modified and the block check will appear to be wrong, and packets will be rejected.

If parity is being used on the communication line, you must inform both KERMITs, so the desired parity can be added to outgoing characters, and stripped from incoming ones. Both KERMITs should be set to the same parity. The specified parity is used both for terminal connection (CONNECT) and file transfer (SEND, RECEIVE, GET).

The choices for SET PARITY are:

NONE (the default) eight data bits and no parity bit.

MARK seven data bits with the parity bit set to one.

SPACE seven data bits with the parity bit set to zero.

EVEN seven data bits with the parity bit set to make the overall parity even.

ODD   seven data bits with the parity bit set to make the overall parity odd.

NONE means no parity processing is done, and the 8th bit of each character can be used for data when transmitting binary files.

If you have set parity to ODD, EVEN, MARK, or SPACE, then advanced versions of KERMIT will request that binary files will be transferred using 8th-bit prefixing. If the KERMIT on the other side knows how to do 8th-bit prefixing, then binary files can be transmitted successfully. If NONE is specified, 8th-bit prefixing will not be requested. (PC DOS KERMIT has 8th-bit prefixing; GEM DOS KERMIT does not. However, GEM DOS KERMIT does have the image mode modifier, described in the next section.)

SET PROMPT

This allows you to change the program's prompt. This is particularly useful if you are using KERMIT to transfer files between two systems of the same kind, in which case you can change the prompts of the KERMIT programs involved to include appropriate distinguishing information.

End of Section 2

GEM DOS KERMIT Commands

This section describes the commands for the GEM DOS implementation of KERMIT. Each command in this section is described in terms of how it differs from its implementation under PC DOS. Unless indicated otherwise, the commands operate as described in Section 2.

When the command form is given, the optional command modifiers are contained within square brackets ([ ]). The command modifiers are listed at the end of this section. Note that the C command has some modifiers that are unique to it. They are listed under the description of the C command.

GEM DOS KERMIT uses the commands in the following table.

Table 3-1. GEM DOS KERMIT Commands

Command	Description
C	Connect terminal emulator.
G	Get files from remote server.
S	Send file
V	Enter remote server.
X	Exit remote KERMIT.

C (CONNECT) COMMAND

The C command initiates the terminal emulator. (The default escape sequence is Ctrl-]C.)

The C command uses the form:

KERMIT c[phe] [parity]

Some of the C command modifiers differ from the modifiers used by

the other commands. The following table describes the modifiers specific to the C command.

Table 3-2. Modifiers Specific to the C Command

Modifier	Description
p	Sets parity. Valid arguments are listed under "P (Parity) Modifier," at the end of this section.
h	Sets half duplex (also known as local echo).
e	Lets you define an escape character other than the default of CNTRL-]c.

When in connect mode, you can toggle the h state by typing <escape-char> followed by h. <escape-char>? will print help on the valid things you can type after an escape character. Any character not on the list is a no-op, and the character is not transmitted.

G (GET) COMMAND

The G command gets files from a server. The G command uses the form:

```
KERMIT g[ip] [parity] file [file2...]
```

Binary files must be transferred in image mode.

A special note is necessary on wildcarding. It is a feature of the run-time library that wildcards on the command line are expanded so that the program need not worry about wildcard expansion. When you get files from a remote server, you want wildcard expansion to take place at the remote end, not the local end. The way around this is to enclose the string in quotes as in the following example:

```
KERMIT g "*.c" "*.h"
```

S (SEND) COMMAND

The S command sends one or more files to a remote KERMIT.

Wildcards are supported; multiple files are supported.

The S command uses the form:

```
KERMIT s[ip] [parity] file [file2 ...]
```

V (SERVER) COMMAND

The V command initiates the server mode. The V command uses the form:

```
KERMIT v[ip] [parity]
```

The server mode does the following:

- Send files – (Respond to a Get command)
- Receive files – (Respond to a Send command)
- Exit – (Respond to Exit command)

Note that you are either in image mode or ASCII mode for the entire server session.

X (EXIT) COMMAND

The X command is the same as the KERMIT BYE command. It causes a remote KERMIT to exit, and logs you off the system at the same time.

The X command uses the form:

```
KERMIT x[p] [parity]
```

I (IMAGE MODE) MODIFIER

The I modifier initiates image mode, which supresses the end–of–line conversion to allow you to transfer binary files. KERMIT marks the end–of–line with a <CR><LF>. When not in image mode, KERMIT

always looks for this sequence and converts it to a host-specific end-of-line. When image mode is initiated, this conversion is turned off. This allows you to send an executable file without having characters dropped out.

If you are transferring files from one GEM DOS system to another, all of the files can be sent in image mode because you have the same file format. This is not true when going to a different operating system.

The following example sends the files FILE1, FILE2, and FILE3 in image mode:

```
KERMIT si filel file2 file3
```


P (PARITY) MODIFIER

The P modifier causes characters transmitted to be encoded according to the particular parity rule. The default is none. Parity is set by specifying the P modifier with one of the following arguments:

    E -- Even
    O -- Odd
    S -- Space
    M -- Mark
    N -- None

End of Section 3

When Things Go Wrong

Connecting two computers can be a tricky business, and many things can go wrong. Before you can transfer files at all, you must first establish terminal communication. But successful terminal connection does not necessarily mean that file transfer will also work. And even when file transfer seems to be working, things can happen to ruin it.

COMMUNICATION LINE PROBLEMS

If you have a version of KERMIT on your microcomputer, but the CONNECT command doesn't seem to work at all, please do the following:

- Make sure all the required physical connections have been made and have not wiggled loose. If you are using a modem, make sure the carrier light is on.

- If you have more than one connector on your micro, make sure you are using the right one.

- Make sure that the port is set to the right communication speed, or <u>baud rate</u>. Use the SHOW command to find out what the current baud rate is. KERMIT has a SET BAUD command if you need to change the baud rate.

- Make sure that the other communication line parameters, like parity, bits per character, handshake, and flow control are set correctly.

You must consult the appropriate manuals for the systems and equipment in question.

If all settings and connections appear to be correct, and communication still does not take place, the fault may be in your modem. Internal modems (that plug into a slot inside the microcomputer chassis) do <u>not</u> work with KERMIT.

KERMIT normally expects to have full control of the communication port. However, it is sometimes the case that some communications equipment controls the line between the two computers on either end. Examples include modems (particularly "smart" modems), port contention or selection units, multiplexers, local networks, and wide-area networks. Such equipment can interfere with the KERMIT file transfer protocol in various ways:

- It can impose <u>parity</u> upon the communication line. This means that the 8th bit of each character is used by the equipment to check for correct transmission. Use of parity will:

    - Cause packet checksums to appear incorrect to the receiver and foil any attempt at file transfer. In most cases, not even the first packet will get through.

    - Prevent the use of the 8th bit for binary file data.

- If terminal connection works but file transfer does not, parity is the most likely culprit. To overcome this impediment, you should find out what parity is being used and inform KERMIT using the SET PARITY command.

- Communications equipment can also interpret certain characters in the data stream as commands rather than passing them along to the other side. For instance, you might find your "smart" modem suddenly disconnecting you and placing a call to Tasmania. The only way to work around such problems is to put the device into "transparent" or "binary" mode. Most communication devices have a way to do this; consult the appropriate manual. In some cases, transparent mode will also cancel the parity processing and allow the use of the 8th bit for data.

THE TRANSFER IS STUCK

There are various ways in which KERMIT file transfers can become stuck, but since many hosts are capable of generating timeout interrupts when input doesn't appear quickly enough, they can usually resend or "NAK" (negatively acknowledge) lost packets. Nevertheless,

if a transfer seems to be stuck, you can type Ctrl-C or Ctrl-Break to simulate a timeout.

The following sections discuss various reasons why a transfer in progress could become stuck. Before examining these, first make sure that you really have a KERMIT on the other end of the line, and you have issued the appropriate command: SEND, RECEIVE, or SERVER. If the remote side is not a server, remember that you must connect back between each transfer and issue a new SEND or RECEIVE command.

THE MICRO IS HUNG

The micro itself sometimes becomes hung for reasons beyond KERMIT's control, such as power fluctuations. If the micro's screen has not been updated for a long time, then the micro may be hung. Try these steps (in the following order):

- Check the connection. Make sure no connectors have wiggled loose from their sockets. If you're using a modem, make sure you still have a carrier signal. Reestablish your connection if you have to.

- Press Ctrl-C to wake the micro up. This should clear up any protocol deadlock.

- If the problem was not a deadlock, restart the micro and then restart KERMIT, You may have to stop and restart KERMIT on the remote host.

THE REMOTE HOST WENT AWAY

If your local system is working but the transfer is hung, maybe the remote host or the remote KERMIT program crashed.

THE DISK IS FULL

If your local floppy disk or remote directory fills up, the KERMIT on the
machine where this occurs will inform you and then terminate the
transfer. You can continue the transfer by repeating the whole
procedure either with a fresh floppy or after cleaning up your
directory.

MESSAGE INTERFERENCE

You may find that file transfers fail occasionally and upredictably. One
explanation could be that terminal messages are being mixed with
your file packet data. These could include system broadcast messages
(like "System is going down in 30 minutes"), messages from other
users ("Hi Fred, what's that KERMIT program you're always running?"),
notifications that you have requested ("It's 7:30, go home!" or "You
have mail from..."). Most KERMIT programs attempt to disable intrusive
messages automatically, but not all can be guaranteed to do so. It
may be necessary for you to "turn off" such messages before starting
KERMIT.

HOST ERRORS

Various error conditions can occur on the remote host that could
effect file transmission. Whenever any such error occurs, the remote
KERMIT normally attempts to send an informative error message to the
local one, and then breaks transmission, putting you back at KERMIT
command level on the local system.

FILE IS GARBAGE

There are certain conditions under which KERMIT can believe it
transferred a file correctly when in fact it did not. The most likely
cause has to do with the tricky business of file attributes, such as text
vs. binary, 7-bit vs. 8-bit, blocked vs. stream, and so forth. Some
systems have their own peculiarities (the VAX is one), and for them
KERMIT has special commands to allow you to specify how a file

should be sent or stored. However, these difficulties usually crop up
only when sending binary files. Textual files should normally present
no problem between any two KERMIT programs.

End of Section 4

Sample KERMIT Sessions

The following describes typical simple file transfers from a PC DOS system to a GEM DOS system, and from a GEM DOS system to a PC DOS system.

PC DOS TO GEM DOS

To transfer a file from a PC DOS system to a GEM DOS system, do the following:

1. On the GEM DOS system, double-click on COMMAND.PRG to go out to the command line interpreter {c}.

2. Put the GEM DOS system into server mode by typing at the command line interpreter:

   {c}kermit v

   or

   {c}kermit vi

   Use v for ASCII files and vi for non-ASCII files. The following banner indicates that you are in KERMIT:

   Atari ST Kermit version 3.0(4) 3/14/85

3. At the PC DOS machine, type the following:

   C>kermit
   IBM-PC Kermit-MS V2.26
   Type ? for help

   Kermit-MS>send filename.ext

   The GEM DOS machine responds with the following:

   Kermit: Receiving FILENAME.EXT

3. When the PC DOS KERMIT notifies you that the sending is completed, type the following:

`Kermit-MS>bye`

This takes the GEM DOS machine out of server mode, as follows:

```
Kermit: Done.
{c}
```

and returns the PC DOS machine to the operating system.

GEM DOS TO PC DOS

To transfer a file from a GEM DOS system to a PC DOS system, do the following:

1. Put the PC DOS machine into server mode:

```
C>kermit
IBM-PC Kermit-MS V2.26
Type ? for help
```

`Kermit-MS>server`

2. At the GEM DOS machine, type the following:

`{c}kermit s filename.ext`

or

`{c}kermit si filename.ext`

Use s for ASCII files and si for non-ASCII files.

3. The GEM DOS shows your progress with the following:

```
Atari ST Kermit version 3.0(4) 3/14/85
Kermit: Sending FILENAME.EXT
Kermit: Done.
{c}
```

4. When you receive the Done message, return to the PC DOS machine, type Ctrl-C to leave server mode, and then the following:

```
Kermit-MS>exit
```

This returns you to the operating system.

End of Section 5

Index

GEM SID-86™

User's Guide

Contents

Contents

4 Assembly Language Syntax for A and L Commands

A GEM SID-86 Error Messages

Tables

GEM SID-86 Operation

GEM SID-86™ is a powerful symbolic debugger for use with PC DOS. It allows users to test and debug programs interactively using symbolic assembly and disassembly, expressions involving hexadecimal, decimal, ASCII, and symbolic values, permanent breakpoints with pass counts, and trace without call.

Note: The distribution disks contain a sample GEM SID-86 session in a file called **SAMPLE.SID**.

INVOKING GEM SID-86

To invoke GEM SID-86, do the following:

1. Start the GEM Desktop™.
2. Open the TOOLS folder.
3. Double-click on the GEMSID.APP icon.

When GEM SID-86 is loaded into memory, it displays the following banner and prompt:

```
---------------------------------------------------------------
GEMSID-86    07 May 1985   *** G E M ***      Version 1.1
Serial No. xxxx-0000-654321          All Rights Reserved
Copyright (C) 1983,1984,1985     Digital Research Inc.
---------------------------------------------------------------

#
```

You can then load a file using the E command (described in Section 3).

COMMAND CONVENTIONS

When GEM SID-86 is ready to accept a command, it prompts you with a pound sign, #. In response, you can type a command line, or a CTRL-C to end the debugging session (see "Terminating GEM SID-86" in this section). A command line can have up to 64 characters, and must be terminated with a carriage return. GEM SID-86 does not process the command line until a carriage return is entered.

The first character (in some cases, the first two characters) of each command line determines the command action. The command character can be followed by one or more arguments, which can be symbolic expressions, filenames, or other information, depending on the command. Arguments are separated from each other by commas or spaces. No spaces are allowed between the command character and the first argument. Note that if the first character of a GEM SID-86 command line is a semicolon (;), the entire line is treated as a comment and ignored. Several commands (G, P, S, T, and U) can be preceded by a minus sign. The effect of the minus sign varies between commands. Section 3, "GEM SID-86 Commands," provides a full explanation of the effect of the minus sign on individual commands.

SPECIFYING A 20-BIT ADDRESS

Most GEM SID-86 commands require one or more addresses as operands. Because the 8086 can address up to 1 megabyte of memory, addresses must be 20-bit values. Enter a 20-bit address, as follows:

    ssss:oooo

The ssss represents an optional 16-bit segment number and oooo is a 16-bit offset. GEM SID-86 combines these values to produce a 20-bit effective address:

      ssss0
    + oooo
      -----
      eeeee

The segment value, ssss, is optional. If you omit the segment value, GEM SID-86 uses a default value appropriate to the command being executed, as described in "E Command" in Section 3.

TERMINATING GEM SID-86

Terminate GEM SID-86 by typing a CTRL-C in response to the # prompt. This returns control to the console handler of the operating system. If you use GEM SID-86 to patch a file, write the file to disk using the W command before exiting GEM SID-86.

OPERATION WITH INTERRUPTS

GEM SID-86 operates in systems with interrupts enabled or disabled, and preserves the interrupt state of the program being executed under GEM SID-86. When GEM SID-86 controls the CPU, either when initially invoked or when regaining control from the program being tested, the condition of the interrupt flag is the same as it was when GEM SID-86 was invoked, except for a few critical regions where interrupts are disabled. While the program being tested has control of the CPU, the user's CPU state, which can be displayed with the X command, determines the state of the interrupt flag.

USING GEM SID-86 WITH MAP FILES

To use GEM SID-86 with compilers and assemblers that produce Microsoft...-format MAP files, you must first convert those MAP files to SYM files using MAP2SYM. The correct syntax for using that utility is as follows:

    MAP2SYM <filename.map> filename.sym

End of Section 1

GEM SID-86 Expressions

An important facility of GEM SID-86 is the ability to reference absolute machine addresses through expressions. Expressions can involve names obtained from the program under test that are included in the "SYM" file produced by RASM-86™ or LINK 86™. Expressions can also consist of literal values in hexadecimal, decimal, or ASCII character string form. You can then combine these values with various operators to provide access to subscripted and indirectly addressed data or program areas. This section describes expressions so that you can incorporate them as command parameters in the individual command forms that follow this section.

LITERAL HEXADECIMAL NUMBERS

GEM SID-86 normally accepts and displays values in hexadecimal. The valid hexadecimal digits consist of the decimal digits 0 through 9 and the hexadecimal digits A, B, C, D, E, and F, corresponding to the decimal values 10 through 15, respectively.

A literal hexadecimal number in GEM SID-86 consists of one or more contiguous hexadecimal digits. If you type four digits, then the leftmost digit is most significant, while the rightmost digit is least significant. If the number contains more than four digits, the rightmost four are taken as significant, and the remaining leftmost digits are discarded. The following examples show the corresponding hexadecimal and decimal values for the given input values:

Input Value	Hexadecimal Value	Decimal Value
1	0001	1
100	0100	256
fffe	FFFE	65534
10000	0000	0
38001	8001	32769

LITERAL DECIMAL NUMBERS

Enter decimal numbers by preceding the number with the # symbol. In this case, the number that follows must consist of one or more decimal digits (0 through 9) with the most significant digit on the left and the least significant digit on the right. Decimal values are padded or truncated according to the rules of hexadecimal numbers, as described above, by converting the decimal number to the equivalent hexadecimal value.

The following input values produce the following internal hexadecimal values:

Input Value	Hexadecimal Value
#9	000
#10	000A
#256	0100
#65535	FFFF
#65545	0009

LITERAL CHARACTER VALUES

GEM SID-86 accepts one or two graphics ASCII characters enclosed in apostrophes as literal values in expressions. Characters remain as typed within the paired apostrophes (that is, no case translation occurs) with the leftmost character treated as the most significant, and the rightmost character treated as least significant. Character strings of length one are padded on the left with zero. Strings of length greater than two are not allowed in expressions, except as described in the S command.

Note that the enclosing apostrophes are not included in the character string, nor are they included in the character count, with one exception: a pair of contiguous apostrophes is reduced to a single apostrophe and included in the string as a normal graphics character.

The following example shows input strings and the hexadecimal values they produce. For these examples, note that uppercase ASCII alphabetic characters begin at the encoded hexadecimal value 41; lowercase alphabetic characters begin at 61; a space is hexadecimal 20; and an apostrophe is encoded as hexadecimal 27.

Input String	Hexadecimal Value
'A'	0041
'AB'	4142
'aA'	6141
''''	0027
''''''	2727
' A'	2041
'A '	4120

REGISTER VALUES

You can use the contents of registers in the CPU state of the program under test in expressions. Simply use the register name wherever a number would normally be valid. For example, if you know that at a certain point in the program the BX register points to a data area you want to see, the following command:

DBX

displays the desired area of memory.

Note that when assembling 8086 instructions using the A command, register names are treated differently than in other expressions. In particular, use of a register name in an assembly language statement entered in the A command refers to the name of a register, and not its contents.

STACK REFERENCES

Elements in the stack can be included in expressions. A caret, ^, refers to the 16-bit value at the top of the stack (pointed to by the SS and SP registers in the user's CPU state). A sequence of n carets refers to the nth 16-bit value on the stack. You can use this feature to set a breakpoint on return from a subroutine, when all that is known is that the return address is on the stack, even though the actual value is not known.

The following commands:

    G,^
    G,^^:^

set breakpoints on return from near and far subroutines, respectively.

SYMBOLIC REFERENCES

Given that a symbol table is present during a GEM SID-86 debugging session, you can reference values associated with symbols through the following three forms of a symbol reference:

    (a)   .s
    (b)   @s
    (c)   =s

where s represents a sequence of one to thirty-one characters that match a symbol in the table.

Form (a) produces the 16-bit value corresponding to the symbol s; that is, the value associated with the symbol in the table. Form (b) produces the 16-bit value contained in the two memory locations given by .s, while form (c) results in the 8-bit value at .s in memory. Note that forms (b) and (c) use the contents of the DS register as the segment component when fetching the 16-bit or 8-bit contents of memory. Suppose, for example, that the input symbol table contains two symbols, and appears as follows:

    0100 GAMMA   0102 DELTA

Further, suppose that memory starting at 0100 in the segment referred to by the DS register contains the following byte data values:

    0100: 02
    0101: 3E
    0102: 4D
    0103: 22

Based on this symbol table and these memory values, the symbol references shown to the left below produce the hexadecimal values

shown to the right below. Recall that 16-bit memory values are stored with the least significant byte first, and thus the word values at 0100 and 0102 are 3E02 and 224D, respectively.

Symbol Reference	Hexadecimal Value
.GAMMA	0100
.DELTA	0102
@GAMMA	3E02
@DELTA	224D
=GAMMA	0002
=DELTA	004D

QUALIFIED SYMBOLS

Duplicate symbols can occur in the symbol table due to separately assembled or compiled modules that independently use the same name for different subroutines or data areas. Further, block structured languages allow nested name definitions that are identical, but non-conflicting. Thus, GEM SID-86 allows reference to "qualified symbols" that take the following form:

S1/S2/ . . . /Sn

The S1 through Sn represents symbols that are present in the table during a particular session.

GEM SID-86 always searches the symbol table from the first to last symbol in the order the symbols appear in the symbol file. For a qualified symbol, GEM SID-86 begins by matching the first S1 symbol, then scans for a match with symbol S2, continuing until symbol Sn is matched. If this search and match procedure is not successful, GEM SID-86 prints the "?" response to the console. Suppose, for example, that the symbol table appears in the symbol file as follows:

0100 A   0300 B   0200 A   3E00 C   20F0 A   0102 A

The memory is initialized as shown in the previous section. In the following example, the unqualified and qualified symbol references produce the hexadecimal values shown.

Symbol Reference	Hexadecimal Value
.A	0100
@A	3E02
.A/A	0200
.C/A/A	0102
=C/A/A	004D
.B/A/A	20F0

OPERATORS IN EXPRESSIONS

Literal numbers, strings, and symbol references can be combined into symbolic expressions using unary and binary "+" and "−" operators. The entire sequence of numbers, symbols, and operators must be written without embedded blanks. Further, the sequence is evaluated from left to right, producing a four-digit hexadecimal value at each step in the evaluation. Overflow and underflow are ignored as the evaluation proceeds. The final value becomes the command parameter, whose interpretation depends on the particular command letter that precedes it.

When placed between two operands, the + indicates addition to the previously accumulated value. The sum becomes the new accumulated value to this point in the evaluation.

The − symbol causes GEM SID-86 to subtract the literal number or symbol reference from the 16-bit value accumulated thus far in the symbolic expression. If the expression begins with a minus sign, then the initial accumulated value is taken as zero. That is,

$$-x$$

is computed as

$$0-x$$

The x is any valid symbolic expression. For example, the following command:

```
DFF00-200,-#512
```

is equivalent to the simple command:

```
DFD00,FE00
```

In commands that specify a range of addresses (B, D, L, F, M and W), the ending address of the range can be indicated as an offset from the starting address. To do this, precede the desired offset by a plus sign. For example, the following command:

```
D121,+7
```

displays the memory from address 121 to 128 (121+7). Use of the unary plus operator at other times is not allowed.

SAMPLE SYMBOLIC EXPRESSIONS

The formulation of GEM SID-86 symbolic expressions is most often closely related to the program structures in the program under test. Suppose you want to debug a sorting program that contains these data items:

LIST: names the base of a table of byte values to sort, assuming there are no more than 255 elements, denoted by LIST(0), LIST(1), ... , LIST(254).

N: is a byte variable that gives the actual number of items in LIST, where the value of N is less than 256. The items to sort are stored in LIST(0) through LIST(N-1).

I: is the byte subscript that indicates the next item to compare in the sorting process. LIST(I) is the next item to place in sequence, where I is in the range 0 through N-1.

Given these data areas, the command

    D.LIST,+#254

displays the entire area reserved for sorting:

    LIST(0), LIST(1), . . . , LIST(254)

The command

    D.LIST,+=I

displays the LIST vector up to and including the next item to sort:

    LIST(0), LIST(1), . . . , LIST(I)

The command

    D.LIST+=I,+0

displays only LIST(I). Finally, the command

    D.LIST,+=N-1

displays only the area of LIST that holds active items to sort:

    LIST(0), LIST(1), . . . , LIST(N-1)

End of Section 2

GEM SID-86 Commands

This section defines GEM SID-86 commands and their arguments. GEM SID-86 commands give you control of program execution and allow you to display and modify system memory and the CPU state.

DEFAULT SEGMENT VALUES

GEM SID-86 has an internal mechanism that keeps track of the current segment value, making segment specification an optional part of a GEM SID-86 command. GEM SID-86 divides the command set into two types of commands:

● The first type pertains to the code segment. These commands, which include the A (Assemble), L (List Mnemonics), P (Pass Points) and W (Write) commands, default to the internal type-1 segment value if no segment value is specified in the command.

● The second type pertains to the data segment. These commands, which include the B (Block Compare), D (Display), F (Fill), M (Move), S (Set), and SR (Search) commands, default to the internal type-2 segment value if no segment value is specified in the command.

When invoked, GEM SID-86 sets both segment values to 0.

A (ASSEMBLE) COMMAND

The A command assembles 8086 mnemonics directly into memory. The A command takes the form

   As

The s is the 20-bit address where assembly starts. GEM SID-86 responds to the A command by displaying the address of the memory location where assembly begins. At this point, you enter assembly language statements as described in Section 4. When a statement is entered, GEM SID-86 converts it to binary, places the value(s) in memory, and displays the address of the next available memory location. This process continues until you enter a blank line or a line containing only a period.

GEM SID-86 responds to invalid statements by displaying a question mark, ?, and redisplaying the current assembly address.

Note that wherever a numeric value is valid in an assembly language statement, an expression can be entered. There is one difference between expressions in assembly language statements and those appearing elsewhere in GEM SID-86. Under the A command, references to registers refer to the names of the registers, while elsewhere they refer to the contents of the registers. Under the A command, there is no way to reference the contents of a register in an expression.

Default Segment Value:

   type-1   Uses default value if no segment value is specified.
               When the A command explicitly specifies a value, sets
               segment value to the value specified.

Examples:

Assemble at offset 213:

    #a213

Set AX register to decimal 128:

    nnnn:0213 mov ax,#128

Push AX register on stack:

    nnnn:0216 push ax

Call procedure whose address is the value of the symbol PROC1:

    nnnn:0217 call .proc1

Test the most significant bit of the byte whose address is the value of
the second occurrence of the symbol I:

    nnnn:021A test byte [.i/i], 80

Jump if zero flag set to the location whose address is the value of the
symbol DONE:

    nnnn:021E jz .done

Move the contents of the memory byte whose address is the value of
the symbol ARRAY plus 4 to the AL register:

    nnnn:0220 mov al,[.array+4]

B (BLOCK COMPARE) COMMAND

The B command compares two blocks of memory and displays any discrepancies at the screen. The B command takes the form:

Bs1,f1,s2

The s1 is the 20-bit address of the start of the first block; fl is the offset of the final byte of the first block, and s2 is the 20-bit address of the start of the second block. If the segment is not specified in s2, the same value is used that was used for s1.

Any differences in the two blocks display at the screen in the form:

al bl a2 b2

The a1 and the a2 are the addresses in the blocks; b1 and b2 are the values at the indicated addresses. If no differences are displayed, the blocks are identical.

Default Segment Value:

    type-2   Uses default value if no segment value is specified. When the B command explicitly specifies a value, sets segment value to the value specified.

Examples:

Compare 200H bytes starting at 40:0 with the block starting at 60:0:

#b40:0,1ff,60:0

Compare 256 byte array starting at offset ARRAY1 in the extra segment with ARRAY2 in the extra segment:

#bes:.array1,+ff,.array2

D (DISPLAY) COMMAND

The D command displays the contents of memory as 8-bit or 16-bit hexadecimal values and in ASCII. The following are the possible forms:

(a) D
(b) Ds
(c) Ds,f
(d) DW
(e) DWs
(f) DWs,f

The s is the 20-bit address where the display starts, and f is the 16-bit offset within the segment specified in s where the display finishes.

Memory is displayed on one or more display lines. Each display line shows the values of up to 16 memory locations. For the first three forms, the display line appears as follows:

    ssss:oooo bb bb . . . bb cc . . . c

The ssss is the segment being displayed, and oooo is the offset within segment ssss. The bb's represent the contents of the memory locations in hexadecimal, and the c's represent the contents of memory in ASCII. A period represents any non-graphics ASCII character.

In response to form (a), GEM SID-86 displays memory from the current display address for 12 display lines. The response to form (b) is similar to form (a), except that the display address is first set to the 20-bit address s. Form (c) displays the memory block between locations s and f. The next three forms are analogous to the first three, except that the contents of memory are displayed as 16-bit values, rather than 8-bit values, as shown:

    ssss:oooo wwww wwww . . . wwww cccc . . . cc

During a long display, you can abort the D command by typing CTRL-BREAK at the console.

Default Segment Value:

   type-2   Uses default value if no segment value is specified. When a D command explicitly specifies a value, sets segment value to the value specified.

Examples:

Display memory bytes from offset F00H through F23H in the current data segment:

   `#df00,f23`

Display 11 bytes starting at the location of ARRAY (I):

   `#d.array+=i,+#10`

Display memory words from offset 80H through FFH:

   `#dw#128,#255`

E (LOAD PROGRAM, SYMBOLS FOR EXECUTION) COMMAND

The E command loads a file into memory so that the next G, T, or U command can begin program execution, and allows one or more symbol table files to be loaded. The E command takes the forms:

(a) E <filename>
(b) E <filename> <symbol filename> {, symbol filename...}
(c) E * <symbol filename> {, symbol filename...}
(d) E
(e) -E (with forms (a) and (b))

Form (a) loads the file by the name <filename>. The file is assumed to be either a .COM or .EXE file as described in the IBM® DOS documentation. If no filetype is specified, .EXE is assumed. To load a .COM file, you must specify the file type with the filename. The contents of the CS, DS, ES, SS, SP, and IP registers are altered according to the type of file loaded. When the load is complete, GEM SID-86 displays the start and end address of the memory block where the file was loaded. Use the V command to redisplay this information at a later time.

Form (b) loads the file <filename> as precedingly described, and then loads one or more symbol table files. If the filetype is omitted from a symbol filename, .SYM is assumed. GEM SID-86 displays the message

    SYMBOLS

when it begins loading the symbol file(s). If GEM SID-86 detects an invalid hex digit or an invalid symbol name, it displays an error message and stops loading the symbol file. The symbols loaded up to the time the error occurred can be displayed with the H command to determine the exact location of the error in the .SYM file. A maximum of 64K bytes is available for symbol table storage.

GEM SID-86 loads symbols at the top of memory, and adjusts the memory size word in the program segment prefix of the loaded program accordingly. Programs often size memory only once during initialization, so we recommend that all symbols be loaded before executing the program being tested. No symbol files should be loaded after program execution begins.

Form (c) does not load a program, but simply loads the indicated symbol table file(s).

Form (d) releases all memory being used by GEM SID-86 for program and/or symbol table storage.

Generally, LINK-86 inserts a code prefix into files it creates, whose function is to set up the environment for the program to execute. Because the user is usually not interested in this prefix, it is automatically executed when the file is loaded using form (a) or form (b). If you want GEM SID-86 not to execute the prefix, you must use form (e). You can then proceed to the point where the prefix transfers control to the main program using these commands:

```
G,102
T
```

When loading a program file with the E command, GEM SID-86 releases any blocks of memory used by any previous E or R commands or by programs executed under GEM SID-86. Thus only one file at a time can be loaded for execution, and that file should be loaded before any symbol tables are read.

GEM SID-86 issues an error message if a file does not exist or cannot be successfully loaded in the available memory.

The format of the symbol table file is that produced by RASM-86 or LINK 86, as follows:

```
      nnnn   symbol1   nnnn   symbol2   ...
        .
        .
        .
        .
```

The nnnn is a four-digit hexadecimal number, and spaces, tabs, carriage returns, and line-feeds can serve as delimiters between hex values and symbol names. Symbol names can be up to thirty-one characters in length.

Default Segment Value:

type-1   When an E command loads a file, sets the segment value
           to the value of the CS register.

type-2   When an E command loads a file, sets the segment value
           to the value of the DS register.

Examples:

Load file TEST.EXE:

```
#etest
```

Load file TEST.EXE and symbol table file TEST.SYM:

```
#etest.exe test.sym
```

Load file TEST.EXE and symbol table files TEST.SYM, IO.SYM and
FORMAT.SYM:

```
#etest test io format
```

Load only the symbol table file TEST1.SYM:

```
#e* testl
```

Debugging Applications with Large Symbols

GEM SID-86 features large symbols that have both a segment and an
offset. To debug GEM applications with large symbols, you might have
to correct the large symbol segment values.

The following forms of the E command ensure the correct large
symbol addresses for the GEM application:

E filename -symfilename
E filename +symfilename

If, after loading the GEM application and symbol file into GEM SID-86,
the symbol addresses are missing the segment value or are otherwise
in error, re-enter the E command using the E filename -symfilename
form. If the symbol addresses are still wrong, use the E filename
+symfilename form. The effects of the minus and plus signs in front
of the symbol table filename are explained below.

Minus Sign

A minus sign (-) in front of the symbol table filename makes the symbols relative to the GEM application's PSP (Program Segment Prefix) plus 10h paragraphs.

When the minus sign is specified in front of the symbol table filename, the SYM file's segment values are constructed by adding 10h paragraphs to the PSP segment value (which is generally the beginning of the code segment). This sum is then added to the code and data segment values provided in the SYM file to generate the correct segment values for the large symbols.

The equations used to construct the code and data segment values for the SYM file can be illustrated as follows:

```
    PSP Segment                  PSP Segment
  + 10h Paragraphs             + 10h Paragraphs
  + SYM file CS                + SYM file DS
  _____             _____
  = Large Symbol CS            = Large Symbol DS
```

Plus Sign

When a plus sign (+) is placed in front of the symbol table filename, the start of the SYM file is offset by the value of the user code segment.

The equations used to construct the code and data segment values for the SYM file can be illustrated as follows:

```
    Current User CS              Current User CS
  + Current SYM file CS        + Current SYM file DS
  _____        _____
  = New SYM file CS            = New SYM file DS
```

F (FILL) COMMAND

The F command fills an area of memory with a byte or word constant. The following are possible forms:

(a) Fs,f,b
(b) FWs,f,w

The s is a 20-bit starting address of the block to be filled, and f is a 16-bit offset of the final byte of the block within the segment specified in s.

In response to form (a), GEM SID-86 stores the 8-bit value b in locations s through f. In form (b), the 16-bit value w is stored in locations s through f in standard form, low 8 bits first followed by high 8 bits.

If s is greater than f or the value b is greater than 255, GEM SID-86 responds with a question mark. GEM SID-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existent RAM at the location indicated.

Default Segment Value:

type-2   Uses default value if no segment value is specified. When an F command explicitly specifies a value, sets the segment value as specified.

Examples:

Fill memory from 100H through 13FH with 0:

```
#f100,13f,0
```

Fill the 256-byte block starting at ARRAY with the constant FFH:

```
#f.array,+255,ff
```

G (GO) COMMAND

The G command transfers control to the program being tested, and optionally sets one or two breakpoints. The following are possible forms:

    (a) G
    (b) G,b1
    (c) G,b1,b2
    (d) Gs
    (e) Gs,b1
    (f) Gs,b1,b2
    (g) -G (with forms a through f)

The s is a 20-bit address where program execution is to start, and b1 and b2 are 20-bit addresses of breakpoints. If no segment value is supplied for any of these three addresses, the segment value defaults to the contents of the CS register.

In forms (a), (b), and (c), no starting address is specified, so GEM SID-86 derives the 20-bit address from the CS and IP registers. Form (a) transfers control to your program without setting any breakpoints. Forms (b) and (c) set one and two breakpoints respectively before passing control to your program. Forms (d), (e), and (f) are analogous to (a), (b), and (c), except that the CS and IP registers are first set to s.

The forms in (g) are analogous to forms (a) through (f), except that intermediate pass point displays are suppressed.

Once control has been transferred to the program under test, it executes sequentially until a breakpoint is encountered. At this point, GEM SID-86 regains control, clears the breakpoints set by the G command, and indicates the address at which execution of the program under test was interrupted, as shown:

```
*ssss:oooo    .symbol
```

The ssss corresponds to the CS, oooo corresponds to the IP where the break occurred, and .symbol is the symbol whose value is equal to oooo, if such a symbol exists. When a breakpoint returns control to GEM SID-86, the instruction at the breakpoint address has not yet been executed.

Default Segment Value:

When GEM SID-86 regains control from a user program after a G command, it sets the type-1 segment value to the new value of the CS register.

Examples:

Begin program execution at address given by CS and IP registers with no breakpoints set:

    #g

Begin program execution at label START in the code segment, setting a breakpoint at label ERROR:

    #g.start,.error

Continue program execution address given by CS and IP registers, with breakpoints at label ERROR and at the address at the top of the stack (the return address of the procedure being executed):

    #g,.error,^

Begin execution with a breakpoint at 34FH, suppressing intermediate pass point display:

    #-g,34f

H (HEXADECIMAL MATH) COMMAND

The H command provides several useful arithmetic functions. The following are the possible forms:

(a) Ha,b
(b) Ha
(c) H

Form (a) computes the sum (ssss), difference (dddd), product (pppppppp), and quotient (qqqq) with the remainder (rrrr) of two 16-bit values. The results are displayed in hexadecimal as follows:

```
+ ssss  - dddd  * pppppppp  / qqqq  (rrrr)
```

Underflow and overflow are ignored in addition and subtraction.

Form (b) displays the value of the expression a in hexadecimal, decimal, ASCII (if the value has a graphic ASCII equivalent), and symbolic (if a symbol exists with a value equal to the value of the expression) forms as shown:

```
hhhh  #ddddd  'c'  .s
```

Form (c) displays the symbols currently loaded in the GEM SID-86 symbol table. Each symbol is displayed in the form:

```
nnnn  <symbol name>
```

You can abort the display by pressing Ctrl-C at the console.

Default Segment Value: None.

Examples:

List all symbols and values loaded with E command(s):

```
#h
```

Show the value of the symbol OPEN in hex and decimal:

```
#h.open
```

Show the word contents of the memory location at INDEX in hex and decimal:

```
#h@index
```

Show sum, difference, product, and quotient of 5C28H and 80H:

```
#h5c28,80
```

I (INPUT COMMAND TAIL) COMMAND

The I command prepares a file control block and command tail buffer in GEM SID-86's program segment prefix, and copies this information into the program segment prefix of the last file loaded with the E command. The command takes the following form:

I<command tail>

The <command tail> is a character string that usually contains one or more filenames. The first filename is parsed into the default file control block at 005CH. The optional second filename is parsed into the second part of the default file control block beginning at 006CH. The characters in <command tail> are also copied into the default command buffer at 0080H. The length of <command tail> is stored at 0080H, followed by the character string terminated with a binary zero.

If a file has been loaded with the E command, GEM SID-86 copies the file control block and command buffer from the program segment prefix of SID- 86 to the program segment prefix of the program loaded. The location of GEM SID-86's program segment prefix can be obtained from the 16-bit values at location 0:6. The location of the program segment prefix of a program loaded with the E command is the value displayed for DS on completion of the program load.

Default Segment Value: None.

Examples:

Set up a file control block at 5CH for the file FILE1.EXE and put the string "file1.exe" in the buffer at 80H (in the program segment prefix of the last file loaded with an E command):

    #ifile1.exe

Set up file control blocks at 5CH and 6CH for the files A:FILE1 and B:FILE2, and copy the string following the i into the buffer at 80H:

    #ia:file1 b:file2 c:file3 $px

L (LIST) COMMAND

The L command lists the contents of memory in assembly language. The following are possible forms:

(a) L
(b) Ls
(c) Ls,f
(d) −L
(e) −Ls
(f) −Ls,f

The s is a 20-bit address where the list starts, and f is a 16-bit offset within the segment specified in s where the list finishes.

Each disassembled instruction is in the form:

label:
ssss:oooo <prefixes> opcode <operands> <.symbol>

The label is the symbol whose value is equal to the offset oooo, if such a symbol exists; <prefixes> are segment override, lock and repeat prefixes; opcode is the mnemonic for the instruction; <operands> field contains 0, 1, or 2 operands, as required by the instruction; and <.symbol> is the symbol whose value is equal to the numeric operand, if there is one and such a symbol exists.

Form (a) lists twelve disassembled instructions from the current list address.   Form (b) sets the list address to s and then lists twelve instructions.   Form (c) lists disassembled code from s through f.   The last three forms are analogous to the first three, except that no symbolic information is displayed (the labels and <.symbol> fields are omitted).

The list address is always set to the next unlisted location in preparation for the next L command.   When GEM SID-86 regains control from a program being tested (see G, T, and U commands), the list address is set to the current value of the CS and IP registers.

You can abort long displays by typing CTRL-BREAK during the list process.   Or, enter CTRL-S to halt the display temporarily.

The syntax of the assembly language statements produced by the L command is described in Section 4.

If the memory location being disassembled is not a valid 8086 instruction, GEM SID-86 displays it in the form:

    ??=   nn

The nn is the hexadecimal value of the contents of the memory location.

Default Segment Value:

   type-1   Uses default value if no segment value is specified. When an L command explicitly specifies a value, sets the segment value to the value specified.

Examples:

Disassemble instructions from 243CH through 244EH:

    #l243c,244e

Disassemble 20H bytes from the label FIND:

    #l.find,+20

Disassemble 12 lines of code from the label ERR plus 3:

    #l.err+3

M (MOVE) COMMAND

The M command moves a block of data values from one area of memory to another. The M command takes the following form:

Ms,f,d

The s is the 20-bit starting address of the block to be moved, f is the offset of the final byte to be moved within the segment described by s, and d is the 20-bit address of the first byte of the area to receive the data. If the segment is not specified in d, the same value is used that was used for s.

Note that if d is between s and f, part of the block being moved is overwritten before it is moved, because data is transferred starting from location s.

Default Segment Value:

> type-2   Uses default value if no segment value is specified. When an M command explicitly specifies a value, sets the segment value to the value specified.

Examples:

Move 10 bytes from 20:2400 to 30:100:

    #m20:2400,+9,30:100

Move 64 bytes from ARRAY to ARRAY2:

    #m.array,+#63,.array2

P (PASS POINT) COMMAND

The P command sets, clears, and displays pass points. The following are the possible forms:

(a) Pa,n
(b) Pa
(c) -Pa
(d) -P
(e) P

A pass point is a permanent breakpoint that remains in effect until it is explicitly removed, as opposed to breakpoints set with the G command, that must be reentered with each G command. Pass points have associated pass counts ranging from 1 to 0FFFFH. The pass count indicates how many times the instruction at the pass point executes before the control returns to the console. Up to sixteen pass points can be set at a time.

An important distinction between breakpoints and pass points is that when execution stops at a breakpoint, the instruction at the breakpoint has not yet been executed. When execution stops because a pass point whose pass count has reached 1, the instruction at the pass point has been executed. This makes it simple to proceed from a pass point with a G command without immediately encountering the same pass point.

Forms (a) and (b) are used to set pass points. Form (a) sets a pass point at address a with a pass count of n, where a is the 20-bit address of the pass point, and n is the pass count, in the range 1 to 0FFFFH. If a pass point is already active at a, the pass count is simply changed to n. GEM SID-86 responds with a question mark if there are already 16 active pass points.

Form (b) sets a pass point at a with a pass count of 1. If a pass point is already active at a, the pass count is simply changed to 1. GEM SID-86 responds with a question mark if there are already 16 active pass points.

Forms (c) and (d) are used to clear pass points. Form (c) clears the pass point at location a. GEM SID-86 responds with a question mark if there is no pass point set at a. Form (d) clears all the pass points.

Form (e) displays all the active pass points in the following form:

    nnnn ssss:oooo .symbol

The nnnn is the current pass count for the pass point, ssss:oooo is the segment and offset of the pass point location, and .symbol is the symbolic name of the offset of the pass point, if such a symbol exists.

When a pass point is encountered, GEM SID-86 displays the pass point information in the following form:

    nnnn PASS ssss:oooo .symbol

The nnnn, ssss:oooo, and .symbol are as precedingly described. Next, GEM SID-86 displays the CPU state before the instruction at the pass point is executed. GEM SID-86 then executes the instruction at the pass point. If the pass count is greater than 1, GEM SID-86 decrements the pass count and transfers control back to the program under test.

When the pass count reaches 1, GEM SID-86 displays the break address (that of the next instruction to be executed) in the following form:

    *ssss:oooo   .symbol

Once the pass count reaches 1, it remains at 1 until the pass point is cleared or the pass count is changed with another P command.

Use pass points with the G, T, and U commands. When you use the G or U command, you can suppress the intermediate pass point display with the – G or –U forms (see G Command and U Command). In this case, only the final pass points (when the pass count = 1) are displayed. You can interrupt the G or U command before its normal termination by pressing CTRL-BREAK at the console. GEM SID-86 aborts the G and U command when the next pass point is encountered and prompts for the next command.

You can also use pass points with breakpoints set with the G command. If a pass point and a breakpoint are set at the same address, the breakpoint is encountered first. Otherwise, pass points behave in the usual manner, decrementing the pass count until it reaches 1 and then returning control to GEM SID-86.

Normally, the segment registers are not displayed at pass points. You can use the S/-S command to enable/disable the segment register display (see S Command).

Default Segment Value:

type-1   Uses this segment default if none is specified.

Examples:

Display active pass points:

    #p

Set pass point at label ERROR:

    #p.error

Set pass point at label PRINT with count of 17H:

    #p.print,17

Clear all pass points:

    #-p

Clear pass point at label ERROR:

    #-p.error

QI, QO (QUERY I/O) COMMAND

The QI and QO commands allow access to any of the 65,536 input/output ports. The QI command reads data from a port; the QO command writes data to a port. The QI command takes the following forms:

    QIn
    QIWn

The n is the 16-bit port number. In the first case, GEM SID-86 displays the 8-bit value read from port n. In the second case, GEM SID-86 displays a 16- bit value from port n.

The QO command takes the following forms:

    QOn,v
    QOWn,v

The n is the 16-bit port number, and v is the value to output. In the first case, the 8-bit value v is written to port n. If v is greater than 255, GEM SID-86 responds with a question mark. In the second case, the 16-bit value v is written to port n.

<u>Default Segment Value</u>: None.

Examples:

Display the 16-bit value of input port 20H:

    #qiw20

Display the 8-bit value of input port 1024:

    #qi#1024

Set the 16-bit output port number 20H to 0FF7EH:

    #qow20,FF7E

Set the 8-bit output port number 1025 to 2:

    #qo#1025,2

R (READ) COMMAND

The R command reads a file into a contiguous block of memory. The R command takes the forms:

    (a) R<filename>
    (b) R<filename>,s

The <filename> is the name and type of the file to be read, and s is the location to which the file is read. Form (a) lets GEM SID-86 determine the memory location into which the file is read. Form (b) causes GEM SID-86 to read the file into the memory segment beginning at s. This address can have the standard form (ssss:oooo) as in the following example:

    #RCPM.SYS,1000:0

The low-order four bits of s are assumed to be zero, so GEM SID-86 reads files on a paragraph boundary.

GEM SID-86 reads the file into memory and displays the start and end addresses of the block of memory occupied by the file. A V command can redisplay this information at a later time. The default display pointer (for subsequent D commands) is set to the start of the block occupied by the file:

The R command does not free any memory used by another R or E command. Thus, many files can be read into memory without overlapping.

GEM SID-86 issues an error message if the file does not exist or there is not enough memory to load the file.

Default Segment Value:

type-1   Uses default value if no segment value is specified.
          When an R command reads a file, GEM SID-86 sets the
          segment value to the base segment where the file was
          read. ·

type-2   Changes this segment default.   When an R command
          reads a file, GEM SID-86 sets the segment value to the
          base segment where the file was read.

Examples:

Read file SID86.COM into memory:

    #rsid86.com

Read file TEST into memory:

    #rtest

Read file TEST into memory starting at location 1000:0:

    #rtest,1000:0

S (SET) COMMAND

The S command changes the contents of bytes or words of memory. The following are possible forms:

(a) Ss
(b) SWs
(c) S
(d) –S

where s is the 20-bit address where the change occurs.

GEM SID-86 displays the memory address and its current contents on the following line. In response to form (a), the display is

   ssss:oooo bb

and in response to form (b), the display is

   ssss:oooo wwww

where bb and wwww are the contents of memory in byte and word formats, respectively.

In response to one of these displays, you can either alter the memory location or leave it unchanged. If you enter a valid expression, the contents of the byte (or word) in memory is replaced with the value of the expression. If you do not enter a value, the contents of memory are unaffected, and the contents of the next address are displayed. In either case, GEM SID-86 continues to display successive memory addresses and values until you enter a period on a line by itself or until GEM SID-86 detects an invalid expression.

In response to form (a), you can enter a string of ASCII characters, beginning with a quotation mark and ending with a carriage return. The characters between the quotation mark and the carriage return are placed in memory starting at the address displayed. No case conversion takes place. The next address displayed is the address following the character string.

GEM SID-86 issues an error message if the value stored in memory cannot be read back successfully, indicating faulty or non-existent RAM at the location indicated.

Forms (c) and (d) control the display of the segment registers when the CPU state is displayed with the trace command and at pass points. Form (c) turns on the segment register display; form (d) turns it off. It is often convenient to turn off the display while debugging to allow the CPU state display to fit on one line.

Default Segment Value:

    type-2   Uses default value if no segment value is specified. When an S command explicitly specifies a value, sets the segment value to the value specified.

Examples:

Begin set at ARRAY (3):

   `#s.array+3`

Set byte to 0:

   `1000:1234  55  0`

Set 3 bytes to 'a', 'b', 'c':

   `1000:1235  55  ''abc`

Set byte to decimal 75:

   `1000:1238  55  #75`

Terminate set command:

   `1000:1239  55  .`

Enable segment register display in CPU state display:

   `#s`

Disable segment register display in CPU state display:

   `#-s`

SR (SEARCH) COMMAND

The SR (Search) command searches a block of memory for a given pattern of numeric or ASCII values and lists the addresses where the pattern occurs. The SR command takes the following form:

    SRs,f,"string"
    Ss,f,value

The s is the 20-bit starting address of the block to be searched, and the f is the offset of the final address of the block.

The SRs,f,"string" form searches for a string of ASCII characters. The "string" parameter specifies the string of one or more printable ASCII characters you want to search for. Note that you may use either single (') or double (") quotes.

The SRs,f,value form searches for a string of numerical characters. The value parameter specifies the string of nonprintable ASCII characters, numbers, and hexadecimal values you want to search for.

For each occurrence of the string or hex value, GEM SID-86 displays the 20-bit address of the first byte of the pattern, in the following form:

    ssss:oooo

If no addresses are listed, the string or value was not found.

Default Segment Value:

    type-2   Uses default value if no segment value is specified.
                When an SR command explicitly specifies a value, sets
                segment value to the value specified.

Examples:

Search the memory block between TEXT1 and TEXT25 for the string "Error":

```
#sr.textl,.text25,"Error"
```

Search the memory block between 0:0 and 0:0FFFFH for the 3-character pattern that starts with "x", ends with "y", and has a decimal 27 in the middle:

```
#sr0:0,ffff,"x",#27,"y"
```

T (TRACE) COMMAND

The T command traces program execution for 1 to 0FFFFH program
steps, displaying the CPU state before each step. The T command
takes the following forms:

(a) T
(b) Tn
(c) TW
(d) TWn
(e) -T (with forms a through d)

The n is the number of program steps to execute before returning
control to the console. If n is omitted, a single program step is
executed.

A program step is generally a single instruction, with the following
exceptions:

● If a DOS interrupt instruction is traced, the entire DOS function is
  treated as one program step, and executes in real time. This is
  because GEM SID-86 itself makes DOS calls, and the DOS is not
  reentrant.

● If the traced instruction is a MOV or POP whose destination is a
  segment register, the CPU executes the next instruction immediately.
  This is because a feature of the 8086 that disables interrupts
  (including the Trace Interrupt) for one instruction after a MOV or
  POP loads a segment register. This allows a sequence like the
  following:

      MOV   SS, STACKSEGMENT
      MOV   SP, STACKOFFSET

  to be executed with no chance of an interrupt occurring between
  the two instructions, at which time the stack is undefined. A
  sequence of such MOV or POP instructions plus one instruction after
  the sequence is considered a one program step.

● If any of the TW forms are used and the traced instruction is a
  CALL, CALLF or INT, the entire called subroutine or interrupt handler
  (and any subroutines called therein) is treated as a one program
  step and executes in real time.

Before each program step is executed, GEM SID-86 displays the CPU state, the disassembled instruction to be executed, the symbolic name of the instruction operand (if any), and the contents of the memory location(s) referenced by the instruction (if appropriate). (See X Command for a detailed description of the CPU state display.) If there is a symbol whose value is equal to the IP, the symbol name followed by a colon is displayed on the line preceding the CPU state display. The segment registers are not normally displayed with the T command, which allows the entire CPU state to be displayed on one line. To enable the segment register display, use the S command (see S Command). With the segment register display enabled, the display of the CPU state is identical to that of the X command.

In all the forms, control transfers to the program under test at the address indicated by the CS and IP registers. If n is not specified, as in form (a), one program step is executed. Otherwise, GEM SID-86 executes n program steps and displays the CPU state before each step, as in form (b). You can abort a long trace before n steps have been executed by typing CTRL-BREAK at the console.

When n steps have been executed, GEM SID-86 displays the address of the next instruction to be executed, along with the symbolic value of the IP, if there is such a symbol, in the following form:

```
*ssss:oooo   .symbol
```

Forms (c) and (d) are analogous to forms (a) and (b), except in the way subroutine calls are treated. In the TW forms, the entire subroutine called from the program level being traced is treated as a single program step, and executes in real time. This allows tracing at a high level of the program, ignoring subroutines that have already been debugged, or for other reasons are not currently of interest.

If the command is preceded by a minus sign, as in form (e), symbolic labels and symbolic operands are omitted from the CPU state display. This can speed up the display when large symbol tables are loaded, by skipping the symbol table lookup.

When a single instruction is being traced, interrupts are disabled for the duration of the instruction. This prevents GEM SID-86 from tracing through interrupt handlers when debugging on systems in which interrupts occur frequently.

After a T command, the list address used in the L command is set to the address of the next instruction to be executed, and the default segment values are set to the CS and DS register values.

Default Segment Value:

    type-1   When GEM SID-86 regains control from a user program after a T command, it sets this segment value to the value of the CS register.

    type-2   When GEM SID-86 regains control from a user program after a T command, it sets this segment value to the value of the DS register.

Examples:

Trace one program step:

    #t

Trace 65535 steps:

    #tffff

Trace 500 program steps with symbolic lookup disabled:

    #-t#500

U (UNTRACE) COMMAND

The U command is similar to the T command except that the CPU state is displayed only before the first instruction is executed, rather than before every step. The following are possible forms:

    (a) U
    (b) Un
    (c) UW
    (d) UWn
    (e) -U (with forms a through d)

The n is the number of instructions to execute before returning control to the console. You can abort the U command before n steps have been executed by striking Ctrl-Break at the console.

Form (e) differs from the analogous T command in that GEM SID-86 disables the display of intermediate pass points (while the pass count is greater than 1). In this case, only when the pass count reaches 1 is the pass information displayed (see P Command).

Default Segment Value:

    type-1   When GEM SID-86 regains control from a user program
              after a U command, it sets this segment value to the
              value of the CS register.

    type-2   When GEM SID-86 regains control from a user program
              after a U command, it sets this segment value to the
              value of the DS register.

Examples:

Trace without display 200H steps:

    #u200

Trace without display 200H steps, suppressing the intermediate pass point display:

    #-u200

V (VALUE) COMMAND

The V command displays the start and end addresses of the block of memory where the last file was loaded with the E or R commands, excluding symbol tables loaded with the E command. The V command takes the following form:

    V

GEM SID-86 responds to the V command with a question mark if neither the R nor E commands have been used.

Default Segment Value: None.

W (WRITE) COMMAND

The W command writes the contents of a contiguous block of memory to disk. The following are the possible forms:

(a) W<filename>
(b) W<filename>,s,f

The <filename> is the filename and filetype of the disk file to receive the data, and s and f are the 20-bit first and last addresses of the block to be written. If the segment is not specified in f, GEM SID-86 uses the same value that was used for s.

With form (a), GEM SID-86 assumes the s and f values from the last file read with an R command. If no file was read with an R command, GEM SID-86 responds with a question mark. This form is useful for writing out files after patches have been installed, assuming the overall length of the file is unchanged.

With form (b), where s and f are specified as 20-bit addresses, the low four bits of s are assumed to be 0. Thus the block being written must always start on a paragraph boundary.

If a file with the name specified in the W command already exists, SID- 86 deletes it before writing a new file.

Default Segment Value:

   type-1   Uses default value if no segment value is specified. Change default if specified explicitly.

Examples:

Write to the file TEST.EXE the contents of the memory block read into by the most recent R command:

```
#wtest.exe
```

Write the contents of the memory block 40:0 through 40:3FFF to the file TEST.EXE on drive B:

```
#wb:test.exe,40:0,3fff
```

X (EXAMINE CPU STATE) COMMAND

The X command allows you to examine and alter the CPU state of the program under test. The X command takes the forms:

  (a) X
  (b) Xr
  (c) Xf

The r is the name of one 8086 CPU register and f is the abbreviation of one CPU flag. Form (a) displays the CPU state in the format:

```
            AX    BX    CX  . . .  SS    ES    IP
--------- xxxx  xxxx  xxxx . . . xxxx  xxxx  xxxx
<instruction>   <symbol name>   <memory value>
```

The nine hyphens at the beginning of the line indicate the state of the nine CPU flags. Each position can be either a hyphen, indicating that the corresponding flag is not set (0), or a 1-character abbreviation of the flag name, indicating that the flag is set (1). This table shows the abbreviations of the flag names.

Table 3-1. Flag Name Abbreviations

Character	Name
O	Overflow
D	Direction
I	Interrupt Enable
T	Trap
S	Sign
Z	Zero
A	Auxiliary Carry
P	Parity
C	Carry

The <instruction> is the disassembled instruction at the next location to be executed, which is indicated by the CS and IP registers. If the

symbol table contains a symbol whose value is equal to one operand in <instruction>, the symbol name is displayed in the <symbol name> field, preceded by a period. If instruction references memory, the contents of the referenced location(s) are displayed in the <memory value> field, preceded by an equal sign. Either a byte, word, or double word value is shown, depending on the instruction. In addition to displaying the machine state, the first form changes the values of the default segments back to the CS and DS register values, and the default offset for the L command to the IP register value.

Form (b) allows you to alter the registers in the CPU state of the program being tested. The r following the X is the name of one 16-bit CPU register. GEM SID-86 responds by displaying the name of the register followed by its current value. If you type a carriage return, the value of the register is not changed. If you type a valid expression, the contents of the register are changed to the value of the expression. In either case, the next register is then displayed. This process continues until you enter a period or an invalid expression, or the last register is displayed.

Form (c) allows you to alter a flag in the CPU state of the program being tested. GEM SID-86 responds by displaying the name of the flag followed by its current state. If you type a carriage return, the state of the flag is not changed. If you type a valid value, the state of the flag is changed to that value. Only one flag can be examined or altered with each Xf command. Set or reset flags by entering a value of 1 or 0.

Default Segment Value:

    type-1   When an X command changes the value of the CS register, GEM SID-86 changes this segment value to the new value of the CS register.

    type-2   When an X command changes the value of the DS register, GEM SID-86 changes this segment value to the new value of the DS register.

Examples:

Change registers starting with BP:

    #xbp

Change BP to hex 2B64:

    BP=1000   2b64

Change SI to decimal 12345:

    SI=2000   #12345

Change DI to value of symbol VAR plus 6:

    DI=0020   var+6

Terminate X command:

    CS=0040

Y COMMAND (OUTPUT TO 1 OR 2 SCREENS)

The Y command controls the graphics-to-text conversions under GEM SID-86. The forms are as follows:

YGE
YGD
YG
YME
YMD
Y

The YGE command enables the graphics image buffer. This command causes the graphics image appearing on the screen to be saved in a buffer before GEM SID-86 switches the screen to text. The YG command is used to recall the graphics image buffer to the screen.

The YGD command disables the graphics image buffer. After entering the YGD command, the graphics image is not saved in the graphics image buffer.

The YG command restores the graphics image to the screen. The YG command only functions if the YGE command has enabled the graphics image buffer. Press any character key to restore the GEM SID-86 text to the screen.

The YME command enables multi-screen mode. If you have two screens connected to your system, the YME command routes GEM SID-86 to one screen and graphics to the other. This is the default if the GEM software is not in memory when GEM SID-86 is loaded.

The YMD command disables the multi-screen mode. Both graphics and text are displayed on a single screen. This is the default if the GEM software is in memory at the time GEM SID-86 is loaded.

The Y command displays the status of the graphics image save/restore buffer and the current screen mode.

? COMMAND (HELP)

The ? command prints a list of available GEM SID-86 commands. The
form is as follows:

?

?? COMMAND (HELP)

The ?? command prints a detailed command list that, in addition to the
GEM SID-86 commands, includes the available command options. The
form is as follows:

??

<div align="center">End of Section 3</div>

Assembly Language Syntax
for A and L Commands

In general, the syntax of the assembly language statements in the A and L commands is standard 8086 assembly language. Several minor exceptions appear here.

- Up to three prefixes (LOCK, repeat, segment override) can appear in one statement, but they all must precede the opcode of the statement. Alternately, a prefix can be entered on a line by itself.

- The distinction between byte and word string instructions is made as follows:

      byte   word

      LODSB   LODSW
      STOSB   STOSW
      SCASB   SCASW
      MOVSB   MOVSW
      CMPSB   CMPSW

- The mnemonics for near and far control transfer instructions are as follows:

- JMPS   JMP   JMPF
             CALL   CALLF
             RET   RETF

- If the operand of a CALLF or JMPF instruction is a 20-bit absolute address, it is entered in the form:

      ssss:oooo

  The ssss is the segment and oooo is the offset of the address.

● Operands that refer to a byte or a word are ambiguous and must be preceded either by the prefix "BYTE" or "WORD", as in the following example:

```
INC     BYTE [BP]
NOT     WORD [1234]
```

Failure to supply a prefix when needed results in an error message. (These prefixes can be abbreviated to "BY" and "WO".)

● Operands that address memory directly are enclosed in square brackets to distinguish them from immediate values, as in the following example:

```
ADD     AX,5        ;add 5 to register AX
ADD     AX,[5]      ;add the contents of
                    ;location 5 to AX
```

● The following are the forms of register indirect memory operands:

```
[pointer register]
[index register]
[pointer register + index register]
```

The pointer registers are BX and BP, and the index registers are SI and DI. Any of these forms can be preceded by a numeric offset, as in the following example:

```
ADD     BX,[BP+SI]
ADD     BX,3[BP+SI]
ADD     BX,1D47[BP+SI]
```

End of Section 4

GEM SID-86 Error Messages

AMBIGUOUS OPERAND: An attempt was made to assemble a command with an ambiguous operand. Precede the operand with the prefix "BYTE" or "WORD".

BAD FILE NAME: A filename in an E, R, or W command is incorrectly specified.

BAD HEX DIGIT: A SYM file being loaded with an E command has an invalid hexadecimal digit.

CANNOT CLOSE: The disk file written by a W command cannot be closed.

DISK READ ERROR: The disk file specified in an R command could not be read properly.

DISK WRITE ERROR: A disk write operation could not be successfully performed during a W command, probably due to a full disk.

INSUFFICIENT MEMORY: There is not enough memory to load the file specified in an R or E command.

NO FILE: The file specified in an R or E command could not be found on the disk.

NO SPACE: There is no space in the directory for the file being written by a W command.

PROGRAM TERMINATED NORMALLY: The program running under GEM SID-86 completed, or was terminated by a CTRL-BREAK.

SYMBOL LENGTH ERROR: A symbol in a SYM file being loaded with an E command has more than thirty-one characters.

SYMBOL TABLE FULL: There is no more space in GEM SID-86's symbol table.

VERIFY ERROR AT s:o: The value placed in memory by a Fill, Set, Move, or Assemble command could not be read back correctly, indicating bad RAM or attempting to write to ROM or non-existent memory at the indicated location.

Index